RITA BONELLI

LA GRANDE GUIDA DEL PROGRAMMATORE

- LE ISTRUZIONI DEL BASIC
- USO DEL VIDEO E DELLA TASTIERA
- GESTIONE PROGRAMMI
- STAMPANTE
- ELEMENTI DI PROGRAMMAZIONE
- I CODICI
- GESTIONE ERRORI
- IL VIDEO E I CARATTERI
- LA TASTIERA
- IL BASIC COMPILATO
- LINGUAGGIO MACCHINA E ASSEMBLER
- I FILE SU CASSETTA
- I FILE SU DISCO
- LA GRAFICA
- ADATTATORE TELEMATICO
- GLI SPRITE • IL SUONO
- VALORI DELLE NOTE
- REGISTRI DEL VIC II
- UTILIZZO MEMORIA E CONFIGURAZIONI
- SISTEMA GEOS

GRUPPO EDITORIALE
JACKSON

commodore 64

LA GRANDE GUIDA

DEL PROGRAMMATORE

RITA BONELLI

GRUPPO
EDITORIALE
JACKSON
Via Rosellini 12
20124 Milano

© Copyright per l'edizione originale:
GRUPPO EDITORIALE JACKSON - S.p.A.

COPERTINA: Emiliano Bernasconi
GRAFICA E IMPAGINAZIONE: Anna Colombo
STAMPA. C.P.M. - Ponte Sesto di Rozzano - MI

Tutti i diritti riservati. Stampato in Italia. Nessuna parte di questo libro può essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dall'editore.

PREFAZIONE

Il COMMODORE 64, questo fortunato home computer è ancora ben presente sul mercato italiano, al punto che è stato approntato e messo in vendita l'ADATTATORE TELEMATICO 6499, per consentire di usarlo come terminale intelligente.

Il responsabile della divisione libri del Gruppo Editoriale Jackson mi ha chiesto di trasformare la mia collana di 3 volumi sul COMMODORE 64 in un unico volume, in modo di fornire ai lettori una piccola e completa enciclopedia che permetta loro di padroneggiare completamente il sistema costituito dal COMMODORE 64, dall'unità nastro, dall'unità a floppy disk, dalla stampante e dall'adattatore telematico.

Inoltre viene illustrato il nuovo sistema operativo GEOS, sistema operativo ad icone, che rende ancora più semplice l'uso del calcolatore.

All'interno dell'Enciclopedia, strumento essenziale per gli utenti del COMMODORE 64, si riportano accanto alla trattazione teorica una numerosa serie di programmi esempio, tutti provati sul calcolatore, che consentono al lettore di imparare ad usare e programmare il suo sistema.

Rita Bonelli

INDICE GENERALE

PARTE I

Capitolo 1 - UNO SGUARDO D'INSIEME I-1
Capitolo 2 - LINGUAGGIO BASIC I-27
Capitolo 3 - COLLOQUIO VIDEO/TASTIERA I-103

PARTE II

Capitolo 1 - MEMORIZZAZIONE E CARICAMENTO PROGRAMMI II-1
Capitolo 2 - I FILE SU STAMPANTE II-19
Capitolo 3 - COSTRUZIONE DI PROGRAMMI II-59
Capitolo 4 - CODICI E NUMERI DEL CALCOLATORE II-87

PARTE III

Capitolo 1 - ERRORI III-1
Capitolo 2 - IL VIDEO E I CARATTERI III-7
Capitolo 3 - LA TASTIERA III-85
Capitolo 4 - IL BASIC COMPILATO III-89

PARTE IV

Capitolo 1 - LA PROGRAMMAZIONE IN ASSEMBLER
E IN LINGUAGGIO MACCHINA

PARTE V

Capitolo 1 - File su disco V-1
Capitolo 2 - File SEQUENZIALI V-19
Capitolo 3 - File RANDOM V-37
Capitolo 4 - File RELATIVI V-87
Capitolo 5 - Errori V-107
Capitolo 6 - Programmi di utilità V-III
APPENDICE A - MAGIC DESK I V-127
APPENDICE B - CALC RESULT V-131
APPENDICE C - Le basi di dati V-133

PARTE VI

Capitolo 1 - LA GRAFICA VI-1
Capitolo 2 - CARATTERI VI-11
Capitolo 3 - PAGINA GRAFICA VI-51
Capitolo 4 - ALTRE POSSIBILITÀ VI-75
Capitolo 5 - UN PICCOLO SISTEMA GRAFICO VI-91

PARTE VII

Capitolo 1 - ADATTATORE TELEMATICO VII-1
Capitolo 2 - GLI SPRITE VII-7
Capitolo 3 - IL SUONO VII-49
Capitolo 4 - I REGISTRI DEL VIC II E I REGISTRI DEL SID VII-91

PARTE VIII

Capitolo 1 - UTILIZZO DELLA MEMORIA E CONFIGURAZIONI VIII-1
Capitolo 2 - IL SISTEMA OPERATIVO GEOS VIII-49

COMMODORE 64
LA GRANDE GUIDA
DEL PROGRAMMATORE

1

INDICE

Capitolo 1 - UNO SGUARDO D'INSIEME
1.1 Il calcolatore 1
1.2 Il Basic 11
1.3 Il programma 17

Capitolo 2 - LINGUAGGIO BASIC
2.1 Introduzione 27
2.2 Costanti e Variabili 30
2.3 Operatori Aritmetici, Relazionali e Logici 34
2.4 Istruzioni 42
ABS 46
ASC 46
ATN 46
CHR$ 47
CLOSE 48
CLR 49
CMD 50
CONT 51
COS 52
DATA 52
DEF FN 53
DIM 54
END 55
EXP 55
FN 55
FOR...TO...STEP 56
FRE 60
GET 61
GET# 62
GOSUB 63
GOTO 65
IF ... THEN 66
INPUT 67
INPUT# 68
INT 69
LEFT$ 70
LEN 71
LET 71

LIST 72
LOAD 72
LOG 74
MID$ 74
NEW 75
NEXT 75
ON 76
OPEN 78
PEEK 80
POKE 80
POS 81
PRINT 81
PRINT# 82
READ 83
REM 84
RESTORE 85
RETURN 85
RIGHT$ 86
RND 86
RUN 90
SAVE 90
SGN 91
SIN 92
SPC 92
SQR 93
STATUS 94
STEP 94
STOP 95
STR$ 95
SYS 96
TAB 96
TAN 97
THEN 97
TIME 97
TIME$ 98
TO 98
USR 98
VAL 99
VERIFY 99
WAIT 100

Capitolo 3 - COLLOQUIO VIDEO/TASTIERA
3.1 I due stati del calcolatore 103
3.2 Uso tasti e caratteri di controllo 103
3.3 Controllo colore 106
3.4 Input controllato 111
3.5 Preparazione maschere video 117

CAPITOLO 1

UNO SGUARDO D'INSIEME

1.1 IL CALCOLATORE

Dopo aver effettuato le operazioni di montaggio suggerite dal manuale contenuto nella scatola ti trovi davanti la tastiera e il video; dai corrente e vedi comparire sul video:

```
**** COMMODORE 64 BASIC V2 ****

64K RAM SYSTEM 38911 BASIC BYTES FREE

READY.
```

con un quadratino lampeggiante sotto la prima lettera della parola READY. Tale quadratino lampeggiante si chiama CURSORE.
Cosa è successo? Il COMMODORE 64 ti avvisa che è pronto per lavorare e attende i tuoi comandi. Vediamo cosa sta sotto questo.

Il calcolatore è una macchina costruita per funzionare in un modo ben definito, una macchina elettronica. Il modo nel quale essa è stata costruita la rende molto versatile, al punto che non sembra solo una macchina.

La parte fisica del calcolatore è il suo HARDWARE. Esso si schematizza nei seguenti componenti:

.. Unità Centrale
.. Unità di Entrata
.. Unità di Uscita.

Figura 1.1 Schema del calcolatore

Nell'Unità Centrale (CPU-Central Processing Unit) si distinguono le seguenti parti:

.. Unità di Controllo
.. Unità Aritmetico Logica
.. Memoria

Figura 1.2 Schema dell'unità centrale del calcolatore

L'Unità Centrale è il calcolatore; esso ha bisogno almeno di una Unità di Entrata e di una Unità di Uscita per comunicare con l'uomo.

Il calcolatore sa fare una sola cosa: eseguire le istruzioni che vengono immagazzinate nella sua memoria; queste istruzioni costituiscono il suo PROGRAMMA di lavoro.

L'UNITA' DI CONTROLLO fa funzionare il calcolatore eseguendo il programma che sta nella MEMORIA e attivando l'UNITA' ARITMETICO LOGICA, quando serve.

I programmi costituiscono il SOFTWARE del calcolatore, quello che lo rende versatile. Infatti i PROGRAMMI possono essere immagazzinati nella memoria uno per volta, scegliendo ogni volta quello che serve.

L'UNITA' DI ENTRATA serve per comunicare con il calcolatore: per far entrare nella memoria i programmi e i dati necessari per lavorare. L'UNITA' DI USCITA serve per ricevere dal calcolatore i risultati dei lavori eseguiti.

Nel tuo COMMODORE 64 l'Unità centrale sta sotto la tastiera che vedi, l'Unità di entrata è la tastiera e l'Unità di uscita è il video che colleghi alla tastiera. Se vuoi puoi collegare al calcolatore anche altre unità di entrata e uscita; ne parleremo più avanti.

La MEMORIA è fondamentale e ce ne occupiamo quel tanto che basta per poter proseguire. La scritta che compare all'accensione del calcolatore ti dice che hai "38911 BASIC BYTES FREE" (38911 BASIC BYTES LIBERI). La parola BYTES è oramai entrata nel gergo informatico e non viene di solito tradotta. Il byte è l'unità elementare di informazione che può stare nella memoria del calcolatore; esso viene usato come unità di misura della memoria.
La memoria del calcolatore è formata da tante caselle (cellette, contenitori); come un grande foglio di carta disegnata a rettangoli. Ogni casella puoi immaginarla come una scatolina rettangolare, ulteriormente suddivisa in 8 piccoli scomparti.

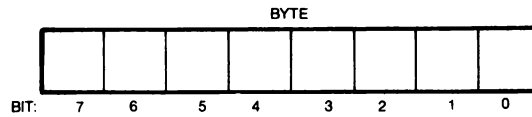


Figura 1.3 Schema di una cella di memoria, con il byte formato da 8 bit

Ognuno degli 8 scomparti può contenere un solo segno, anzi una cifra, che può essere o zero o uno. Il contenuto di ogni scomparto si chiama BIT. Un BYTE è formato da 8 BIT. Il BIT è l'unità minima di informazione per il calcolatore, però non possiamo somministrargli un singolo bit, siamo obbligati a fornirgliene 8 tutti insieme, un intero byte. Se immaginiamo la memoria del calcolatore come un grande foglio di carta, lo vediamo pieno di zeri e di uno.
All'inizio il calcolatore ti dice che hai a disposizione 38911 cellette; puoi quindi sbizzarrirti a scrivere 38911x8 cifre 0 o 1. Devi imparare a riempire le cellette in modo che quello che ci scrivi abbia un significato per il calcolatore.

Dal momento che tu sei abituato a scrivere numeri, parole, simboli per risolvere dei problemi, deve esistere un modo per comunicarli al calcolatore usando solo degli zeri e degli uno. Si tratta ovviamente di usare un codice, nel quale ogni ottetto di zeri e uno abbia un diverso significato.
Combinando in tutti i modi possibili 2 simboli (0 e 1) in gruppi di 8, si ottengono 256 disposizioni diverse. In conseguenza in un byte si possono rappresentare 256 cose diverse; le chiamiamo caratteri. Nel nostro caso si tratta del CODICE ASCII; esistono anche altri tipi di codici.

Nel secondo Volume Capitolo 4 Paragrafo 4 , sono riportati i codici ASCII; potrai vedere che:

00110000 corrisponde alla cifra 0
01000001 corrisponde alla lettera A.

In realtà nella tabella ivi riportata, prodotta da un programma, compaiono solo i codici stampabili (cioè quelli che corrispondono a un vero carattere). Nella tabella il codice ASCII è riportato come numero decimale, esadecimale e binario, vicino al carattere corrispondente. Inoltre, per ogni carattere stampabile, è riportato il codice usato per farlo apparire sul video; codice che risulta diverso dal codice ASCII e viene chiamato D/CODE (display code). Vediamo associato alla cifra zero il numero decimale 48 e alla lettera A il numero decimale 65. Abitualmente si fa riferimento ai codici decimali ASCII, la rappresentazione binaria risulta fastidiosa e quella esadecimale non troppo agevole.

Per il calcolatore si è dovuto usare un'aritmetica diversa da quella decimale, la binaria. In essa valgono le stesse regole dell'aritmetica decimale riguardo al valore posizionale delle cifre in un numero, solo che le cifre disponibili sono due invece che dieci, e il valore posizionale delle cifre si calcola in base alle potenze di due invece che alle potenze di dieci.
Nel nostro BYTE i pesi da attribuire alle singole cifre (BIT) sono:

	BYTE							
BIT:	7	6	5	4	3	2	1	0
PESI:	128	64	32	16	8	4	2	1

Figura 1.4 Valori posizionali dei bit in un byte

che corrispondono, partendo dal bit più a destra, alle seguenti potenze di 2:

$2^0 = 1 \qquad 2^1 = 2 \qquad 2^2 = 4 \qquad 2^3 = 8$

$2^4 = 16 \qquad 2^5 = 32 \qquad 2^6 = 64 \qquad 2^7 = 128$

Ne consegue che il numero più piccolo che si può scrivere in un byte è zero, mentre il più grande è 255; infatti:

$$0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 0$$

$$1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 =$$

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

Possiamo, per il momento, concludere che in un byte si può rappresentare un numero che va da 0 a 255, e che ognuno di questi 256 numeri può essere interpretato come un carattere.

Esiste il problema di distinguere una parte della memoria del calcolatore da un'altra, cioè di individuare i singoli bytes che la compongono.
Questo problema è stato risolto nel modo più semplice possibile; è stato assegnato ad ogni byte un indirizzo numerico. L'assegnazione degli indirizzi è stata fatta partendo da zero, secondo la successione dei numeri naturali.
Nel nome del tuo calcolatore è riportato il numero 64 perchè esso ha una memoria formata da 64K bytes. Dove K è una costante usata come abbreviazione per misurare la memoria del calcolatore; essa corrisponde al numero 1024. In conseguenza il tuo calcolatore ha una memoria formata da 64x1024 byte, cioè da 65536 bytes. Gli indirizzi con i quali vengono individuati singolarmente i bytes vanno da 0 (zero) a 65535. In informatica si comincia sempre a contare da zero, questo serve a risparmiare un bit. La quantità di memoria gestibile da un calcolatore dipende dalle caratteristiche del microprocessore che costituisce la sua struttura portante, in particolare dalla capacità di indirizzamento che esso ha. Nel nostro caso il microprocessore può gestire indirizzi il cui valore massimo è 65535. Per formare gli indirizzi vengono usati due bytes; il primo fornisce l'indirizzo della PAGINA di memoria, il secondo l'indirizzo del BYTE nella pagina. Sono disponibili 256 pagine di 256 bytes ciascuna (256x256=65536).

A questo punto, tornando alla scritta iniziale, viene subito la domanda: dove sono andati a finire parte dei 65536 bytes di memoria, dato che ne restano liberi solo 38911?

Prima abbiamo definito il calcolatore come una macchina che sa solo eseguire le istruzioni che sono immagazzinate nella sua memoria. Vediamo dunque in cosa consiste il funzionamento automatico del calcolatore. Per poter esporre brevemente questo argomento dobbiamo chiarire alcuni termini. Si chiama:

REGISTRO, una zona di memoria, formata da uno o da due bytes consecutivi, nella quale sta memorizzato un numero. Se il registro consiste in un solo byte, esso può contenere un numero che va da 0 a 255. Se esso consiste in due bytes, il valore numerico contenuto si ottiene sommando il contenuto del primo byte al contenuto del secondo moltiplicato 256. In questo secondo caso il numero massimo contenuto nel registro può arrivare a 255 + 255x256 = 65535.

CONTATORE DEL PROGRAMMA (PC - Program Counter), un registro che contiene sempre l'indirizzo del primo byte dell'istruzione di programma che deve essere eseguita.

REGISTRO DELL'ISTRUZIONE, un registro nel quale viene trasferita, prelevandola dalla memoria, l'istruzione che deve essere eseguita.

ACCUMULATORE, un registro che viene usato per effettuare le operazioni aritmetiche.

FLAG, un registro o una parte di registro (anche solo un BIT) che viene usato per ricordare il tipo di risultato di una operazione svolta.

Questi registri, insieme ad altri, stanno nella Unità Centrale del Calcolatore. L'automatismo del calcolatore consiste in questo:

1) viene prelevata dalla memoria l'istruzione il cui indirizzo sta nel Contatore del Programma e trasferita nel Registro dell'Istruzione;

2) il Contatore del Programma viene incrementato in modo da contenere l'indirizzo della successiva istruzione (le istruzioni sono immagazzinate in memoria una di seguito all'altra per indirizzi crescenti);

3) viene eseguita l'istruzione, con tutte le implicazioni che essa comporta, tipo prelevamento di dati dalla memoria e simili;

4) si torna al punto 1).

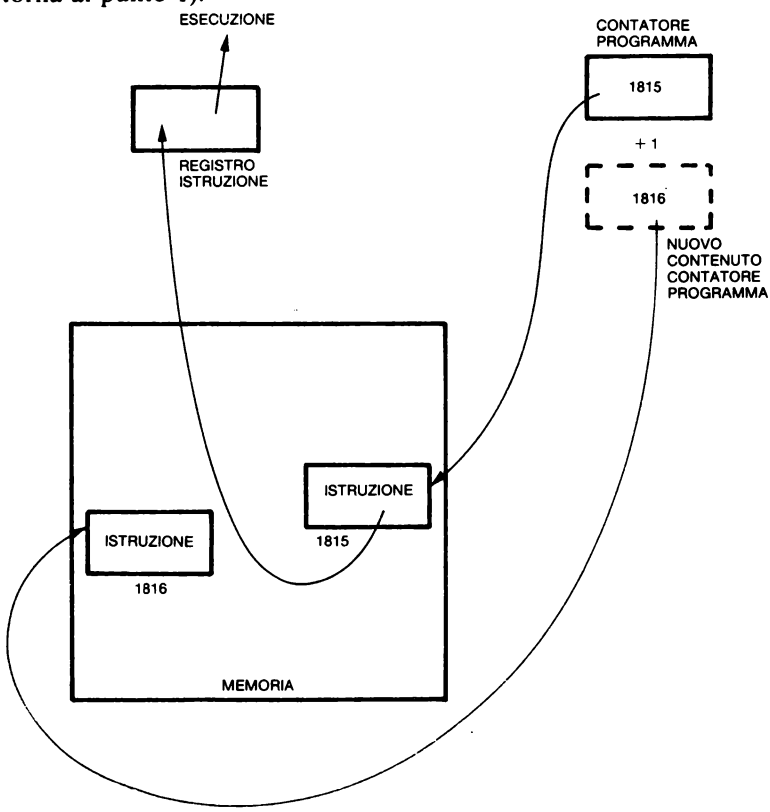


Figura 1.5 Schema del funzionamento automatico del calcolatore

Da quanto detto sembrerebbe che un programma per il calcolatore possa solo procedere per indirizzi crescenti; questo non è vero, dal momento che esistono delle istruzioni capaci di modificare il contenuto del Registro Contatore del Programma.

Per far partire il calcolatore si deve quindi scrivere nel Contatore del Programma un numero, l'indirizzo della prima istruzione da eseguire, dopo aver naturalmente memorizzato il programma, e dare lo START. Per raggiungere questo scopo ci deve essere qualche bottone, qualche tasto, che ci consenta di accedere al Contatore del Programma e alla memoria e un dispositivo per la partenza.
Guardando il COMMODORE 64 tu non vedi queste cose; quando dai corrente appare la scritta iniziale. Questo significa che il tasto di accensione avvia lo svolgimento automatico delle operazioni prima descritte, ponendo nel PC l'indirizzo fisso di un programma che sta già in memoria e viene mandato in esecuzione. Il tuo calcolatore è già in parte programmato e tu comunichi con esso con l'aiuto di un programma. In realtà si tratta di parecchi programmi che possono essere raggruppati con il nome di SISTEMA OPERATIVO (OS - Operating System) e

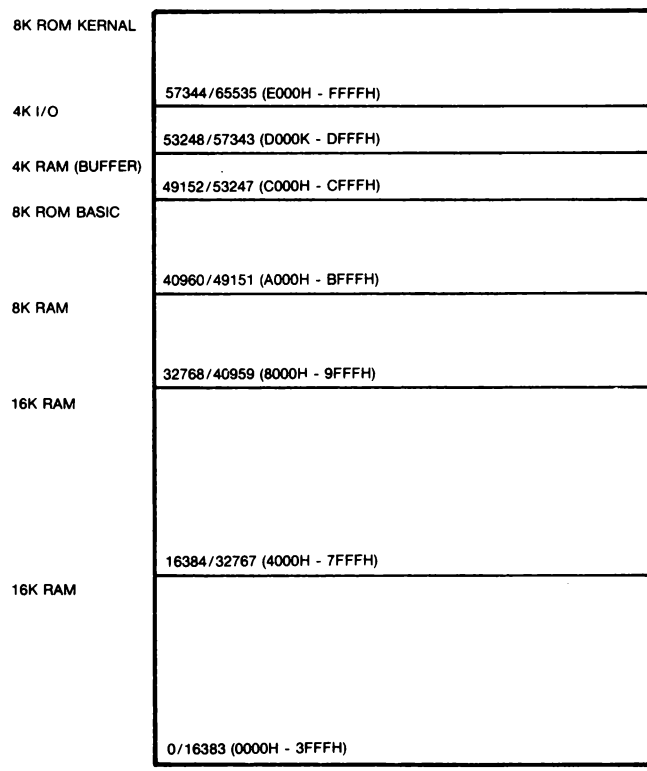


Figura 1.6 Schema dell'utilizzo della memoria al momento dell'accensione.

suddivisi in tre gruppi:

.. GESTIONE VIDEO (SCREEN EDITOR)
.. GESTIONE SISTEMA (ROUTINE KERNAL)
.. INTERPRETE BASIC (BASIC INTERPRETER).

In generale si parla solo di Sistema Operativo facendo riferimento ai primi due gruppi, e di Interprete Basic. Nel seguito chiameremo EDITOR il programma di gestione del colloquio video/tastiera.
Tutti questi programmi sono registrati in modo permanente in una parte della memoria del tuo calcolatore; essa prende il nome di Memoria ROM (Read Only Memory - Memoria a sola lettura). Ecco perchè la scritta iniziale dice che sono disponibili solo 38911 bytes e compare la parola BASIC a ricordare che il tuo calcolatore è pronto a ricevere da te comandi accettabili dall'Interprete Basic.

Dalla figura 1.6 puoi vedere l'utilizzo della memoria al momento dell'accensione del calcolatore. Nel Capitolo 1 dell'ottavo Volume si tratta diffusamente questo argomento; infatti il COMMODORE 64 può utilizzare la memoria in modo diverso da quello qui presentato. Per il momento facciamo riferimento allo schema in figura e precisiamo che nel primo blocco di RAM, quello corrispondente agli indirizzi più bassi, sono utilizzati i primi 2K di memoria per i seguenti scopi:

da 0 a 1023 (0000H-03FFH) memoria di lavoro per Sistema Operativo e Interprete Basic;

da 1024 a 2023 (0400H-07E7) 1000 byte di MAPPA VIDEO;

da 2024 a 2047 (07E8H-07FFH) 24 byte per la grafica (sprite).

Per il programma BASIC dell'utente sono disponibili i 38911 byte che vanno da 2048 a 40959 (0800H-9FFFH).
Da 40960 a 49151 (A000H-BFFFH) si trova la ROM dedicata all'Interprete BASIC.
Da 49152 a 53247 (C000H-CFFFH) sono situati 4K di RAM, utilizzati per i BUFFER.
Da 53248 a 57343 (D000H-DFFFH) è mappato l'I/O.
Da 57344 a 65535 (E000H-FFFFH) sono presenti gli 8K di ROM dedicati alle routine del Sistema Operativo.

Perchè si costruiscono così i Personal Computer? Per renderne semplice l'uso da parte della gente. Facciamo un paragone: per accendere la luce in casa si usa un interruttore, non ci piacerebbe ogni volta dover stabilire i contatti tra i fili elettrici, sarebbe noioso e anche pericoloso. Analogamente sarebbe noioso dover programmare un Personal senza gli intermediari Sistema Operativo e Interprete Basic;

sarebbe anche pericoloso per le case costruttrici, che ne venderebbero pochi, e per noi che non li useremmo, mentre essi sono degli utilissimi strumenti.

Consideriamo con attenzione la figura 1.6; in essa sono schematizzate le diverse parti della memoria e il loro utilizzo. La sigla ROM è già stata spiegata; la sigla RAM significa: Random Access Memory - Memoria ad accesso random, con il significato che in questa parte della memoria si può sia leggere che scrivere (cancellando quello che vi era scritto prima). In realtà si può accedere in modo casuale a tutti gli indirizzi della memoria del calcolatore, sia ROM che RAM. Per questa ragione la sigla più corretta per quest'ultima parte della memoria è RWM, per Read Write Memory (memoria per lettura e scrittura).
Il Sistema Operativo e l'Interprete Basic non possono essere cancellati dall'utente, essi sono formati da diversi programmi che vanno in esecuzione con il procedimento automatico esposto precedentemente. Tutti questi programmi necessitano di una parte di memoria RAM, cioè di memoria nella quale si può anche scrivere, per poter lavorare, questa parte di memoria si trova da 0 a 1023.

I 38911 bytes che vengono inizialmente annunciati come liberi per il Basic sono quanto resta di memoria RAM all'utente per poter lavorare. Tu imparerai a scrivere programmi nel linguaggio BASIC, li caricherai nella memoria RAM libera e l'Interprete BASIC, sempre presente in ROM li manderà in esecuzione, servendosi di quelle parti del Sistema Operativo che via via risultino necessarie.

Prima di parlare del BASIC è bene rendersi conto, almeno in prima approssimazione, della natura di quelle istruzioni di programma di cui si è parlato a proposito dell'automatismo del calcolatore. Ogni microprocessore (e quindi ogni calcolatore) ha un gruppo (SET) di istruzioni che lo fanno funzionare e che costituiscono il suo linguaggio, quello che viene chiamato LINGUAGGIO MACCHINA. Esse sono dei codici binari, cioè dei gruppi di bit, ognuno con il suo significato. Una istruzione è formata da un codice, detto CODICE OPERATIVO, che la definisce, seguito da nessuno, uno o più operandi a seconda dei casi. In generale le istruzioni occupano uno o più byte e la lunghezza dell'istruzione o è implicitamente definita dal codice o viene riportata nel corpo dell'istruzione stessa. Durante il funzionamento automatico del calcolatore il registro Contatore del Programma si incrementa di uno se la istruzione occupa un byte, di N se la istruzione occupa N bytes.

E' possibile suddividere il SET di istruzioni in categorie; esse sono:

.. Istruzioni di Trasferimento, per trasferire contenuti di memoria da un posto all'altro;
.. Istruzioni di Calcolo, per eseguire calcoli;
.. Istruzioni di Confronto, per confrontare tra loro contenuti di memoria;
.. Istruzioni di Scelta, per far procedere il programma da un indirizzo o da un altro a seconda dello stato di un FLAG;

.. Istruzioni di Salto, per far proseguire il programma da un nuovo indirizzo, invece che in sequenza;
.. Istruzioni di Lettura, per ricevere dati o programmi dall'esterno;
.. Istruzioni di Scrittura, per mandare informazioni all'esterno.

In una istruzione di trasferimento gli operandi sono l'indirizzo di memoria da cui prelevare e quello in cui memorizzare. In una istruzione di calcolo, per esempio di somma, gli operandi sono i due addendi. In una operazione di salto l'operando è l'indirizzo dal quale deve proseguire il programma.

Le istruzioni lavorano sui registri della Unità Centrale e sulla memoria; i riferimenti si ottengono mediante gli indirizzi.

Dopo che una istruzione è stata prelevata dalla memoria e portata nella CPU nel Registro della Istruzione, essa viene controllata. Se il codice operativo non viene riconosciuto, cioè non fa parte del linguaggio macchina del microprocessore, il calcolatore si ferma e segnala un errore; infatti esso non sa cosa deve fare.

Il Sistema Operativo e l'Interprete Basic sono scritti in linguaggio macchina, ma tu non hai bisogno di imparare il linguaggio macchina per usare con profitto il tuo calcolatore. Basta imparare le regole d'uso del Sistema Operativo e dell'Interprete Basic.

Torniamo al CURSORE LAMPEGGIANTE che appare sotto la, ormai ben nota, scritta iniziale.

Nel tuo calcolatore sta funzionando il programma EDITOR; esso è in attesa di LEGGERE i tasti che tu premi sulla tastiera. Prova a scrivere:

CIAO COMMODORE 64

e a premere il tasto con sopra scritto RETURN. Vedi apparire:

?SYNTAX ERROR
READY.

Infatti il programma che sta funzionando, quando tu premi il tasto RETURN, comincia ad analizzare la frase che tu hai scritto e confronta i suoi primi caratteri con le parole contenute in una tabella nella memoria, la tabella dei comandi validi; in questa tabella non è presente alcun comando che inizi con "CIAO". Il programma allora ti segnala con "?SYNTAX ERROR" che hai commesso un errore (errore di sintassi) e con "READY." che si è messo di nuovo in paziente attesa di un tuo comando.

In realtà sono successe delle cose abbastanza complicate. Quando tu premi un tasto viene ricevuto un segnale; esso viene interpretato dal programma, riconosciuto come un certo carattere e stampato sul video nella posizione dove prima stava il CURSORE, con conseguente spostamento di questo. Solo quando tu premi il tasto RETURN il messaggio scritto e rivisto sul video viene preso in considerazione dal programma. Inoltre, tu mentre scrivi puoi sbagliare e ti è possibile correggere; questo significa che ogni carattere prima di essere stampato sul video viene analizzato e alcuni caratteri particolari producono uno specifico effetto.
I caratteri battuti sulla tastiera vanno a finire in una zona dedicata di memoria, chiamata BUFFER DI TASTIERA, che ne può contenere fino a dieci. Il buffer è gestito da un registro PUNTATORE, che segnala la posizione libera. Al momento dell'accensione del calcolatore il puntatore individua la prima posizione del buffer.
Il calcolatore si può trovare nello stato di attesa di un tuo carattere battuto, o può essere impegnato in altro lavoro; in quest'ultimo caso il carattere va nel buffer e viene utilizzato appena il calcolatore ha terminato il precedente lavoro.

1.2 IL BASIC

Il BASIC è un linguaggio per scrivere programmi per il calcolatore; la sigla sta per: Beginners All-purpose Symbolic Instruction Code (Linguaggio di programmazione generale per principianti).
Un programma scritto in linguaggio BASIC viene caricato nella memoria del calcolatore e interpretato da un programma già precedentemente presente nella stessa memoria, l'Interprete BASIC. Per l'Interprete Basic il programma scritto in linguaggio Basic costituisce un insieme di dati su cui lavorare.

Il linguaggio ha un suo dizionario formato da un numero limitato di parole e simboli che l'interprete riconosce.
Le parole del dizionario, nella maggior parte dei casi una sola, a volte più di una, sono le istruzioni per il PROGRAMMA INTERPRETE BASIC.
I dati su cui operare, numeri o parole, sono forniti rispettando alcune regole semplici.

Abbiamo visto nel paragrafo precedente che il calcolatore ci segnala con il cursore lampeggiante che è in attesa di comandi, ma che non tutte le parole gli vanno bene. Se proviamo a scrivere: LIST e poi premiamo RETURN. Vediamo scendere il cursore e fermarsi dopo aver scritto: READY.
LIST è un comando; esso viene accettato alla pressione del tasto RETURN ed eseguito. L'effetto del comando è di produrre sul video la lista delle istruzioni del programma BASIC presente in memoria; se in memoria non ci sono istruzioni, esse non compaiono e vediamo solo la parola READY che segnala di aver eseguito l'ordine.

Nel seguito useremo indifferentemente i termini comando e istruzione.

Ci sono due modi per far lavorare il nostro calcolatore:

1) eseguire istruzioni scritte sul video, senza conservarle dopo averle eseguite, cioè lavorare in MODO IMMEDIATO;

2) eseguire istruzioni dopo averle memorizzate, cioè lavorare in MODO DIFFERITO.

Per lavorare nel modo 1), scriviamo una istruzione e la mandiamo in esecuzione quando premiamo il tasto RETURN. Se vogliamo eseguire la stessa istruzione un'altra volta, dobbiamo riscriverla. Per esempio, prova a scrivere:

PRINT "ciao commodore 64" e poi RETURN

vedrai apparire sul video sotto la tua istruzione:

ciao commodore 64

e poi il solito READY.

Se vuoi rifare la stessa cosa devi riscrivere la tua istruzione. Stai lavorando in modo immediato. Quello che tu scrivi resta visibile sul video, ma non è stato conservato nella parte di memoria dedicata al programma. Se continui a scrivere, il quadro video si sposta verso l'alto e le prime cose scritte scompaiono.

Per lavorare nel modo 2) scrivi invece:

10 PRINT "ciao commodore 64" e poi RETURN

alla pressione del tasto RETURN non succede alcunchè, tu vedi sul video solo la tua linea di programma. Prova ora a scrivere:

RUN e poi RETURN

vedrai comparire:

ciao commodore 64

hai cioè eseguito il tuo programma memorizzato (programma di una sola riga); hai lavorato in modo differito, ma per fare eseguire il tuo programma hai dato il comando RUN in modo immediato. Il comando RUN fa eseguire i programmi che sono nella memoria.

Se ora vuoi eseguire tante volte il tuo programma devi solo continuare a dare il comando RUN.
Se provi ora con il comando LIST, ottieni la lista della tua riga di programma.

Da quanto precede possiamo trarre questa conclusione: il sistema calcolatore decide che stiamo scrivendo istruzioni da eseguire in modo differito se esse iniziano con un numero.

Il nostro calcolatore può dunque lavorare in due modi; esso accetta quello che scriviamo quando premiamo il tasto RETURN. In quel momento analizza l'inizio della frase scritta; se inizia con un numero la conserva nella memoria del programma, se inizia con una parola e se questa è un comando valido, lo esegue.
I due modi possono essere mescolati e non si hanno inconvenienti.

Per memorizzare un programma devi scrivere le istruzioni una linea dopo l'altra numerandole progressivamente; devi stare attento a numerarle in modo ordinato. Infatti esse vengono conservate, e poi eseguite, in ordine di numero di linea crescente, anche se tu non sei obbligato a scriverle in ordine. Se scrivi prima la linea 40, poi la linea 30 e poi la linea 10, quando dai il comando LIST esse appaiono in ordine: 10, 30, 40. Questo naturalmente è molto comodo, dal momento che scrivendo si può dimenticare qualcosa. Inoltre tieni presente che, se scrivi due numeri di linea uguali, l'ultima linea scritta va a sostituirsi a quella scritta precedentemente con lo stesso numero.

A questo punto crediamo sia utile, se non lo hai ancora fatto, che tu prenda confidenza con la tastiera e il video; infatti essi sono i principali mezzi per comunicare con il calcolatore.

Prendiamo in considerazione quei tasti che ci permettono di muoverci sul video, cioè di spostare il cursore.
In alto a destra c'è un tasto con scritto CLR/HOME; se lo premi da solo vedi che il cursore va nell'angolo in alto a sinistra del video; hai eseguito il comando HOME, e il video ha mantenuto il suo precedente contenuto. Se invece premi il tasto CLR/HOME insieme a SHIFT, il video viene pulito e il cursore va nell'angolo in alto a sinistra; hai eseguito il comando CLEAR (abbreviato da CLR) insieme al comando HOME.

Quando avvengono queste cose, sta lavorando l'EDITOR; esso gestisce le operazioni tastiera/video, analizza il tasto (o i tasti premuti contemporaneamente) e agisce in conseguenza.

Consideriamo ora i due tasti situati in basso verso destra con le scritte CRSR accompagnate da frecce bidirezionali verticali e orizzontali; essi servono a spostare il cursore sul video. Quello con le frecce verticali, usato da solo fa spostare verso il

basso, usato insieme al tasto SHIFT fa spostare verso l'alto. Quello con le frecce orizzontali, usato da solo fa spostare verso destra, usato insieme al tasto SHIFT fa spostare verso sinistra. Inoltre questi due tasti producono una azione continua se vengono mantenuti premuti; hanno cioè la funzione di ripetizione (repeat) automatica.

In alto a sinistra c'è un tasto con la scritta INST/DEL; esso ha questa funzione: se usato da solo cancella il carattere prima del cursore e fa tornare indietro il cursore. Se non ci sono caratteri scritti fa solo tornare indietro il cursore. Anche esso ha la funzione di ripetizione automatica; nel tornare indietro passa alla riga precedente, quando arriva al limite sinistro del video, fino al raggiungimento della posizione HOME. Se il tasto viene usato insieme a SHIFT, esso, se il cursore si trova in posizione opportuna, provoca lo spostamento verso destra di quello che sta scritto dopo il cursore e crea uno spazio. Il tasto INST/DEL insieme a SHIFT attiva la funzione di INSerT. Non si può inserire dopo l'ultimo carattere scritto, inoltre non si può inserire superando le lunghezze limite consentite, per esempio, per le istruzioni di programma due linee del video. Anche per la funzione di INST si ha la ripetizione automatica mantenendo premuti i due tasti.
Le due funzioni DEL e INST sono di enorme utilità quando si scrive; esse consentono di correggere.

Anche la barra di spaziatura presente sulla tastiera al centro ha la funzione di ripetizione automatica.

Vediamo qualche utile esempio di uso dei tasti appena esaminati. Supponiamo di scrivere un comando in modo immediato, come:

```
PRINT "sto provando i tasti di controllo"
```

quando premi RETURN appare sotto il comando:

una linea bianca (non scritta), la frase tra virgolette nel comando PRINT, una linea bianca e il solito READY.
Prova a portarti con il cursore, usando i tasti freccia, alla fine della frase, appena prodotta dal comando PRINT, e a cancellarla con il tasto DEL. Poi risali con il cursore in modo da posizionarti in un punto qualunque della linea dove è ancora scritto il comando PRINT e premi RETURN. Vedrai di nuovo la frase che hai appena cancellato, infatti per effetto del RETURN mentre il cursore è posizionato sulla riga dell'istruzione, questa viene eseguita di nuovo.

Ora prova a premere contemporaneamente i due tasti che si trovano in basso a sinistra, cioè quello con sopra il marchio COMMODORE e lo SHIFT, vedrai cambiare le scritte che sono sul video; esse passano all'altro set di caratteri. Se premi ancora contemporaneamente i due tasti ritorni alla situazione precedente del video.

Ora se vuoi puoi fare pulizia con CLR/HOME.

Prova ora a premere contemporaneamente i tasti con la scritta RUN/STOP (a sinistra) e RESTORE (a destra); il video viene pulito, compare READY. nella seconda riga e il cursore appare sotto. In realtà la pressione di questi due tasti ripristina le condizioni del calcolatore al momento dell'accensione, senza però cancellare un eventuale programma presente in memoria.

Puoi ora provare tutti i tasti, con o senza il tasto SHIFT. Un modo semplice per provare i tasti è scrivere in modo immediato il comando PRINT, subito dopo premere SHIFT e 2, per aprire le virgolette, e poi battere i tasti voluti. Per ottenere la stampa si devono chiudere le virgolette e premere RETURN. Attenzione però a non superare due linee video! Puoi anche provare a non chiudere le virgolette prima di usare RETURN; ottieni lo stesso la stampa senza segnalazione di errore.

Vediamo ora dove vanno a finire le istruzioni di programma da eseguire in modo differito. Nella mappa di memoria presentata in figura 1.6, abbiamo visto che la parte di memoria RAM a disposizione dell'utente va da 0 a 40959, anzi, dato che i primi 2K sono occupati dalle variabili del sistema e dalla MAPPA VIDEO, l'area disponibile va da 2048 a 40959. Le istruzioni del programma Basic che tu scrivi vengono memorizzate a partire dal byte di indirizzo 2049.
Considerando la memoria sudddivisa in pagine, le variabili del sistema stanno nelle prime 4 pagine: 0, 1, 2 e 3, la memoria dedicata al video sta nelle pagine 4, 5, 6 e 7, il programma Basic comincia in pagina 8. Nel Capitolo 1 dell'ottavo Volume trattiamo diffusamente dell'utilizzo della memoria, per ora ci basta dire che le istruzioni sono memorizzate carattere per carattere a partire dal byte 2049.
Esse sono memorizzate mantenendole in ordine di numero di linea crescente. Questo significa che se tu scrivi per prima l'istruzione con numero di linea uguale a 100, essa viene memorizzata a partire dal byte 2049, se poi scrivi l'istruzione con numero di linea 50, quella con numero di linea 100 viene traslata in memoria verso i byte di indirizzo più alto e lascia libero il posto a partire dal byte 2049 per l'istruzione 50. Questo procedimento viene ripetuto tutte le volte che è necessario mentre tu scrivi un programma. Diciamo che, se tu scrivi le istruzioni in ordine, il sistema lavora di meno! Lo stesso procedimento di spostare verso gli indirizzi più alti parte di un programma avviene anche tutte le volte che si va a correggere una istruzione già scritta aggiungendo qualcosa che si era dimenticato. Analogamente, quando si cancella qualcosa, parte del programma viene traslata verso gli indirizzi più bassi. In conclusione, mentre tu scrivi il programma Basic, il sistema svolge un bel pò di lavoro per tenere in ordine in memoria le tue istruzioni. Quando hai terminato di scrivere il programma una parte della memoria a partire dal byte 2049 è occupata.

Nella zona di memoria usata come area di lavoro dal sistema ci sono dei byte che

servono per riassumere la situazione della zona di memoria dedicata al programma Basic. Per il momento consideriamo i byte di indirizzo 43 e 44 che costituiscono il PUNTATORE all'inizio del programma Basic. Quando si accende il calcolatore l'indirizzo puntato da questi due byte è 2049. Più avanti vedremo come si fa a LEGGERE il contenuto dei byte di memoria. Un altro puntatore che ci interessa considerare ora è quello costituito dai byte 45 e 46. Esso è il puntatore all'inizio delle VARIABILI. Quando si accende il calcolatore l'indirizzo puntato da questi due byte è 2051, cioè l'indirizzo di inizio del programma + 2. Man mano che si introducono le istruzioni del programma l'indirizzo puntato dai byte 45 e 46 si modifica, esso è sempre superiore di 2 all'indirizzo dell'ultimo byte occupato dal programma. Abbiamo usato la parola VARIABILI, con essa si intende definire l'insieme dei dati su cui lavora il programma. Nello scrivere il programma si creano dei nomi simbolici, con regole ben definite che vedremo, che servono ad individuare e distinguere tra loro i dati. Le variabili sono i contenitori dei dati. Ad ogni variabile (contenitore) viene assegnato uno spazio di memoria; l'assegnazione avviene partendo dalla memoria libera subito dopo il programma.

Facciamo un esempio; scriviamo un programma che ci chiede due numeri, definiti simbolicamente come A e B, calcola la somma dei due numeri, chiamandola simbolicamente C, e poi la stampa sul video. Siamo costretti a introdurre la parola chiave del linguaggio che serve per LEGGERE dalla tastiera un dato, essa è INPUT; quella per scrivere l'abbiamo già incontrata, essa è PRINT. Il programma è il seguente:

```
10 INPUT A
20 INPUT B
30 C=A+B
40 PRINT C
```

esso è formato da quattro linee:

.. la linea 10 legge un numero e lo mette nella variabile A;
.. la linea 20 legge un numero e lo mette nella variabile B;
.. la linea 30 esegue la somma del contenuto di A e del contenuto di B e pone il risultato nella variabile C;
.. la linea 40 stampa il risultato della somma contenuto nella variabile C.

Dopo aver scritto il programma il puntatore all'inizio delle variabili ha un valore che dipende dalla lunghezza del programma. Se ora scrivi RUN e poi RETURN, cioè mandi in esecuzione il programma, mentre il programma viene eseguito vengono create le variabili che servono, cioè viene assegnato uno spazio in memoria prima ad A, poi a B e poi a C. Alla linea 10 viene creato lo spazio per A, alla linea 20 viene creato lo spazio per B, alla linea 30 viene ricercata A in memoria dato che serve come addendo, viene cercata B in memoria, dato che serve come addendo, e

viene crato lo spazio per C che serve per contenere il risultato della operazione di somma. Alla linea 40 viene ricercato C in memoria per mandarne il contenuto sul video con il comando PRINT. Le variabili, cioè il loro nome simbolico di riconoscimento ed il loro contenuto attuale, sono memorizzate una dopo l'altra, byte dopo byte, a partire dall'indirizzo di inizio della zona variabili. Tale indirizzo è PUNTATO dai due byte di pagina 0, 45 e 46.

Nell'esempio di programma, sopra riportato, sono scritte 4 linee, numerate 10, 20, 30 e 40. Si usa l'accorgimento di numerare le linee di programma non con numeri consecutivi per facilitare eventuali inserzioni di nuove linee. Se si procede di 10 in 10 sono possibili 9 inserzioni. Nello stesso esempio si vede che le linee di programma 10, 20 e 40 iniziano, dopo il numero di linea, con una parola chiave, un comando Basic, e dopo di questa sono citati dei nomi simbolici, usati per individuare i dati su cui operare. La struttura delle istruzioni Basic è spesso di questo tipo. La linea 30, invece, inizia con il nome di una variabile che deve ricevere il risultato di un calcolo riportato dopo il simbolo uguale. In realtà il linguaggio ha una parola chiave anche per questo tipo di operazioni, che si chiama di assegnazione (variabile = espressione), tale parola è LET, ma è facoltativa e quindi può essere omessa.

Possiamo per il momento concludere che le ISTRUZIONI BASIC sono formate da PAROLE CHIAVE e da OPERANDI. Esse devono essere precedute da un numero di linea per essere eseguite in modo differito.

Ovviamente le PAROLE CHIAVE non devono essere usate come operandi. Nel Capitolo 2 prendiamo in esame le caratteritiche del linguaggio frase per frase. Possiamo parlare indifferentemente di frasi del linguaggio o di istruzioni del linguaggio.

Riteniamo utile avere ben presenti gli argomenti trattati in questo paragrafo, prima di proseguire, questo per studiare il linguaggio Basic cercando di capire cosa concretamente succede nella memoria del calcolatore.

1.3 IL PROGRAMMA

Il programma è una sequenza ordinata di istruzioni. Un problema può essere risolto scrivendo un programma per calcolatore. Non è però il programma che risolve il problema; il problema lo risolve la persona che lo studia, trova il metodo risolutivo adatto ed è capace di tradurre quest'ultimo in un programma per calcolatore. Non è detto che tutte queste cose debbano essere fatte dalla stessa persona. E' però importante notare che il metodo risolutivo di un problema può essere diverso, a seconda che si eseguano, per esempio, dei calcoli in modo manuale, o che si programmino i calcoli per essere eseguiti da un calcolatore elettronico.

Schematizziamo le fasi necessarie per risolvere un problema con il calcolatore:

1) studio del problema;

2) ricerca di un metodo risolutivo e stesura di uno schema che consenta di avere una visione completa di quello che si deve fare;

3) traduzione dello schema in un programma per il calcolatore.

Tu sai che un programma per calcolatore può essere scritto in uno dei tanti linguaggi disponibili; ogni linguaggio ha le sue caratteristiche. La fase 1) e la fase 2) non dipendono dal particolare linguaggio che viene usato nella fase 3); però è meglio aver presente anche nelle prime due fasi su quale calcolatore e con quale linguaggio si vuole lavorare. Questo consente di procedere nelle condizioni ideali per raggiungere risultati ottimali.

La fase 1) e la fase 2) vengono abitualmente chiamate ANALISI del problema. Di solito si parte con un'idea, tipo mi piacerebbe tenere la mia contabilità con il calcolatore, oppure, se uno è appassionato del gioco del Bridge, vorrei usare il calcolatore quando organizzo un torneo tra i miei amici, o altro.

Si deve arrivare a uno schema che metta in evidenza:

1a) quali sono i dati che devono essere elaborati;

2a) quali sono i risultati che si vogliono ottenere.

A questo punto si entra nella fase 2) e si deve arrivare a decidere come fare. I risultati si possono raggiungere in diversi modi e non è facile scegliere il migliore. In questa fase deve anche essere studiato il modo nel quale presentare i dati da elaborare; questo modo può portare a modifiche nei metodi di elaborazione. Dobbiamo onestamente riconoscere che non è molto facile, soprattutto per i principianti, se il problema non è semplice, superare le prime due fasi.

Quello che ti raccomandiamo è di scrivere in qualche modo quello che pensi; è molto importante imparare a prendere appunti ordinati che consentano di rivedere i ragionamenti fatti.

Quando un problema deve essere risolto servendosi di un calcolatore si deve prevedere a priori tutto; il calcolatore infatti non ha il colpo di genio che gli consenta di uscire da un impiccio. Se lo abbiamo programmato male e lo mandiamo in un vicolo cieco, non sa uscirne!

I calcolatori Personal si presentano in modo molto amichevole agli utenti e rendo-

no la programmazione più accessibile che non i grossi calcolatori, però per programmarli bene ci vuole sempre un certo impegno. Molti pensano di poter programmare sedendosi davanti alla tastiera e improvvisando; non è detto che non riescano a produrre qualcosa, magari anche di buono a volte. Noi comunque riteniamo che sia meglio mettersi davanti alla tastiera avendo vicino dei fogli di carta che riportino, il più chiaramente possibile, degli schemi di quello che vogliamo fare.

Esistono delle metodologie collaudate per schematizzare i metodi risolutivi di un problema, prima di arrivare alla stesura del codice nel linguaggio scelto per il calcolatore. Esse si basano sulla tecnica dei diagrammi a blocchi; cioè sulla stesura di disegni formati da simboli grafici diversi collegati da segmenti orientati. Ogni simbolo grafico ha di per sè un preciso significato che facilita la lettura dello schema. Inoltre, all'interno di ogni simbolo si pongono delle frasi esplicative delle operazioni simboleggiate.

I metodi risolutivi vengono spesso chiamati ALGORITMI; si dice che i diagrammi a blocchi sono la descrizione grafica degli algoritmi.

Si possono anche descrivere verbalmente gli algoritmi. Ognuno può scegliere la strada che gli risulta più congeniale.

La visualizzazione mediante schemi di una serie di operazioni da svolgere può essere usata in molte situazioni che non hanno attinenza con l'ambiente della programmazione. Di solito se ne trae un vantaggio di chiarezza e si riescono a vedere le cose in modo sintetico.

In ambiente informatico si chiama DIAGRAMMA DI FLUSSO uno schema grafico che riguarda il movimento di informazioni intorno al calcolatore. Il più semplice diagramma di flusso è quello riportato nella figura 1.7; esso può essere valido in molti casi.

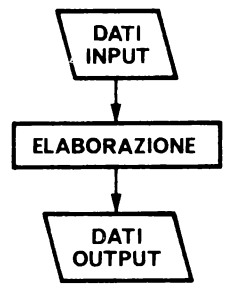


Figura 1.7 Diagramma di flusso

Usiamo la definizione di DIAGRAMMA A BLOCCHI per lo schema grafico che rappresenta la trasformazione dei dati all'interno del calcolatore, cioè quello che fa il programma.

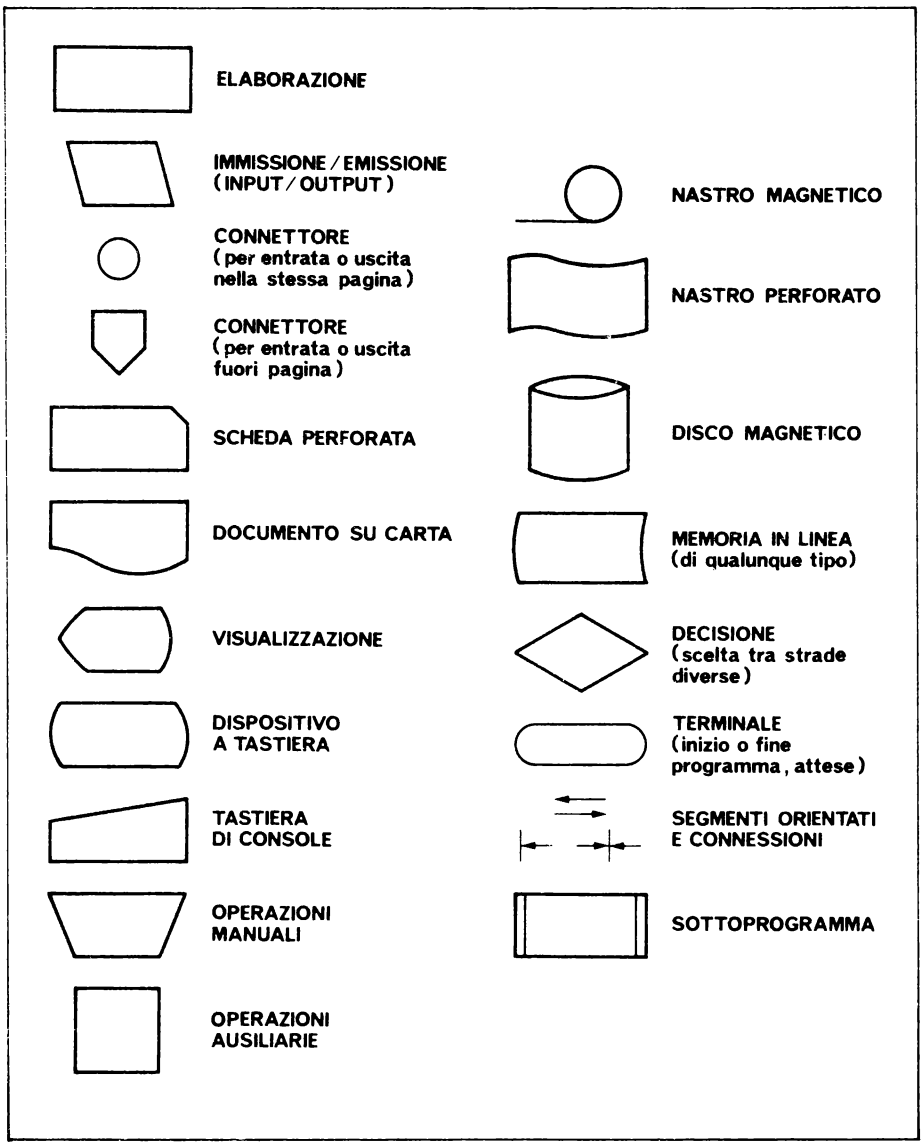


Figura 1.8 Simboli grafici per diagrammi a blocchi

Nella figura 1.8 sono riportati i simboli grafici più comunemente usati per la stesura di diagrammi a blocchi. All'interno dei simboli si scrivono delle indicazioni che personalizzano l'operazione indicata per lo specifico caso.

Ora ci proponiamo di impostare un programma che risolva il seguente problema:

CALCOLARE IL VALORE DI UN POLINOMIO IN X:

$$PL = A(n)X^{n} + A(n-1)X^{n-1} + \ldots + A(1)X + A(0)$$

FACENDO VARIARE:

.. IL GRADO n;
.. GLI n+1 COEFFICIENTI A(k);
.. LA VARIABILE X.

Vogliamo arrivare a scrivere un programma in Basic per il COMMODORE 64.

Il problema è posto in modo sufficientemente chiaro. Precisiamo i dati di INPUT per il programma; essi sono:

.. il grado N del polinomio PL (numero intero);
.. gli N+1 coefficienti A(k) (numeri reali (decimali));
.. il valore della variabile X (numero reale).

Dobbiamo stabilire se vogliamo avere un programma che, fissato N e gli N + 1 coefficienti A(k), possa fornirci il valore di un determinato PL per diversi valori della variabile X, oppure se vogliamo cambiare tutto ogni volta. Ci sembra meglio produrre un programma che ci consenta di percorrere le due strade; cioè, dopo aver calcolato il valore di PL la prima volta, il programma ci chiede se vogliamo cambiare solo X o tutto. A seconda della risposta viene percorsa una o l'altra strada.

La formula (espressione) che ci mostra il polinomio non risulta comoda per il calcolo; infatti si dovrebbero calcolare N potenze della variabile X e non possiamo farlo con un comodo ciclo. Scriviamo PL in un altro modo, esponiamo cioè un algoritmo che si presta ad essere calcolato in modo iterativo (ciclico).

$$PL=((((XA(n)+A(n-1))X+A(n-2))X+\ldots)X+A(1))X+A(0)$$

Con questo algoritmo il calcolo procede così (per maggior chiarezza usiamo

l'asterisco per indicare l'operazione di moltiplica):

```
PL=X*A(n)+A(n-1)
PL=PL*X+A(n-2)
PL=PL*X+A(n-3)

...
PL=PL*X+A(1)
PL=PL*X+A(0)
```

Il risultato del programma deve essere un messaggio che comunichi il valore di PL.

Passiamo ora a descrivere in modo verbale le operazioni necessarie per risolvere il problema posto.

1) Richiesta del grado N e sua introduzione;

2) Analisi del grado N; se N=0 STOP;

3) Impostazione di un ciclo per leggere gli N+1 coefficienti del polinomio e loro introduzione;

4) Richiesta del valore della variabile X e sua introduzione;

5) Calcolo ciclico del valore del polinomio;

6) Stampa del risultato;

7) Domanda circa il proseguimento del programma con lo stesso polinomio e ricezione della risposta;

8) Scelta percorso in base alla risposta; se essa è di proseguire con lo stesso polinomio si va al punto 4); se no si va al punto 1).

La precedente descrizione del programma è a GRANDI BLOCCHI; ogni punto può essere ulteriormente sviluppato arrivando alla sequenza delle singole istruzioni. E' stato necessario introdurre il punto 2) per poter uscire dal programma quando non si desidera più calcolare valori di polinomi.

Segue la descrizione grafica del programma con un diagramma a blocchi che rispetta la precedente descrizione verbale. Dal confronto potrai stabilire quale metodo preferisci.

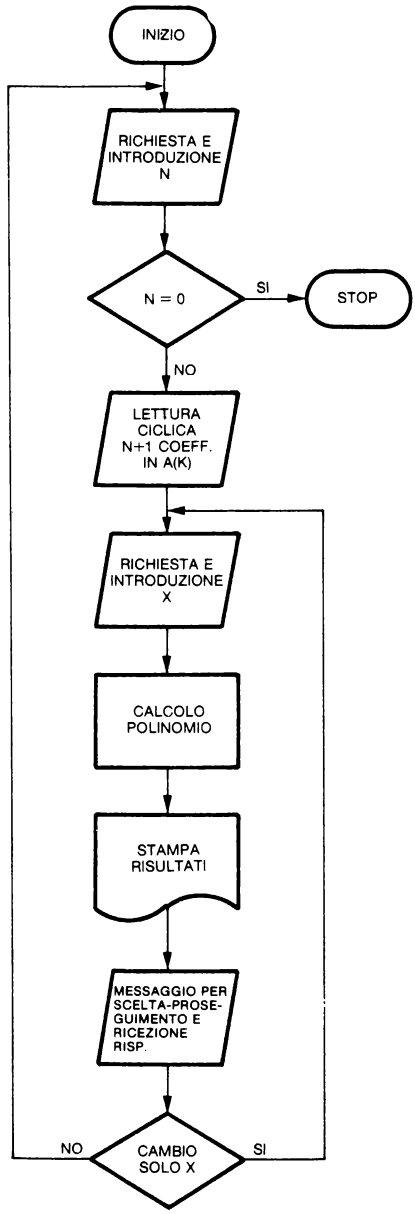


Figura 1.9 Diagramma a blocchi algoritmo calcolo valore polinomio

Dai due schemi riportati si può scendere a schemi più vicini al programma, precisando i singoli passi, ma preferiamo lasciare a te questo lavoro come utile esercizio.

Anche se per i principianti è troppo presto, facciamo seguire il listato del programma in Basic e qualche risultato su stampante. Potrai sempre tornare a questo punto più tardi.

```
1 REM CALCOLO POLINOMIO
10 OPEN4,4:PRINT#4,"     CALCOLO VALORE POLINOMIO"
20 CLOSE4:OPEN4,4
23 INPUT"▯▯▯GRADO POLINOMIO: N%=";N%:PRINT"▯"
25 IFN%=0THENCLOSE4:STOP
30 DIMA(N%):GOSUB100:REM SOTTOPROGRAMMA LETTURA
35 INPUT"▯▯VALORE X: ";X
40 PRINT#4:PRINT#4,"COEFFICIENTI DEL POLINOMIO:"
45 I=0:FORK=N%TO0STEP-1
50 PRINT#4,"A(";K;")=";A(K);"  ";:I=I+1
55 IFI=4THENPRINT#4:I=0
60 NEXTK:IFI<>0THENPRINT#4
65 PRINT#4,"PER X=";X;"IL VALORE DI PL E': ";
70 GOSUB200:PRINT#4,PL:PRINT#4
75 PRINT"▯▯VUOI USARE LO STESSO POLINOMIO "
77 INPUT"RISPONDI S/N ";R$
80 IFR$="S"THEN35
85 RUN20
100 FORK=0TON%
105 PRINT"A(";K;")=";:INPUTA(K)
110 NEXTK:RETURN
200 PL=A(N%)
201 FORK=N%-1TO0STEP-1
203 PL=PL*X+A(K)
205 NEXTK:RETURN
```

RISULTATI SU STAMPANTE

```
    CALCOLO VALORE POLINOMIO

COEFFICIENTI DEL POLINOMIO:
A( 5 )= 1   A( 4 )= 8   A( 3 )=-5   A( 2 )= 3
A( 1 )=-45   A( 0 )= 90
PER X= 3 IL VALORE DI PL E':  738
```

```
COEFFICIENTI DEL POLINOMIO:
A( 5 )= 1    A( 4 )= 8    A( 3 )=-5    A( 2 )= 3
A( 1 )=-45    A( 0 )= 90
PER X= 1 IL VALORE DI PL E': 52
```

Nello sviluppare il programma in Basic abbiamo usato la tecnica dei sottoprogrammi, cioè la chiamata di piccole sequenze di programma che svolgono un determinato compito. Seguono i diagrammi a blocchi dei due sottoprogrammi usati.

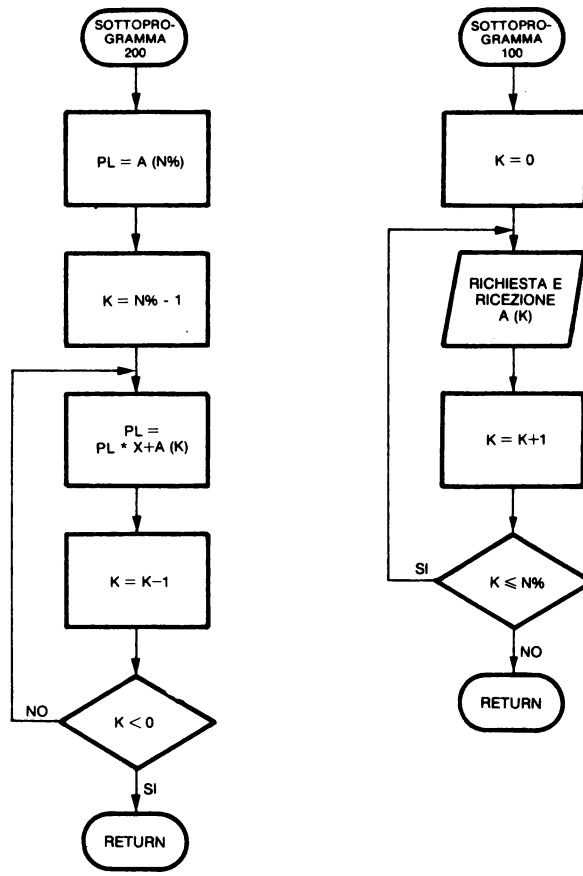


Figura 1.10 Diagrammi a blocchi dei due sottoprogrammi

Alla linea 30 abbiamo potuto usare l'istruzione DIM con N% come dimensione, cioè dimensione variabile, infatti il Basic del COMMODORE 64 lo consente. Con altro calcolatore che non consenta i dimensionamenti variabili si dovrebbe dare un dimensionamento fisso, per esempio a 100, e , dopo la lettura del grado N% del polinomio, controllare se N% < = 100.
L'uscita sulla stampante viene controllata in modo da stampare al massimo 4 coefficienti per riga; usiamo un contatore I all'interno del ciclo di stampa.
L'istruzione RUN20 alla linea 85 serve per abolire il dimensionamento, però RUN cancella anche la tabellina dei file aperti, abbiamo quindi dovuto chiudere alla linea 20 con CLOSE4 la stampante e poi riaprirla.

CAPITOLO 2

LINGUAGGIO BASIC

2.1 INTRODUZIONE

Il linguaggio Basic è di tipo:

.. interpretativo,
.. conversazionale.

La parola INTERPRETATIVO significa che un programma scritto in linguaggio Basic viene eseguito per mezzo di un altro programma che è presente in memoria contemporaneamente. Questo programma si chiama INTERPRETE BASIC e nel COMMODORE 64 si trova memorizzato in modo permanente in 8K di ROM.
La parola CONVERSAZIONALE significa che mentre tu lavori puoi interagire con il sistema calcolatore, intervenendo per modificare il programma in corso o per controllare risultati intermedi o per correggere errori che ti sono stati segnalati dal sistema. Un programma Basic in esecuzione può essere interrotto con la pressione del tasto RUN/STOP; si può intervenire con comandi eseguiti in modo immediato senza danneggiare il programma, e poi proseguire nell'esecuzione con il comando CONT.
Queste due caratteristiche del linguaggio sono quelle che lo rendono di facile apprendimento e che ne hanno consentito una così larga diffusione.
Nella memoria del calcolatore è anche sempre presente il Sistema Operativo, che occupa altri 8K di ROM, le cui routine vengono continuamente usate per procedere nello svolgimento del programma Basic. Le routine del Sistema Operativo sono chiamate KERNAL.
Il programma Basic è formato da una sequenza di frasi o linee. Ogni linea di programma, ad esecuzione differita, inizia con un numero di linea che può andare da 0 a 63999.
Una linea di programma può essere formata da più di una istruzione usando come carattere separatore tra due istruzioni il carattere ":" (due punti). Una linea di programma non può superare 80 caratteri.
Ogni istruzione inizia con una parola chiave, salvo una istruzione, quella di assegnazione, per la quale la parola chiave LET è facoltativa. In generale ogni parola chiave corrisponde a un comando, salvo pochi casi nei quali un comando è formato da più di una parola chiave.

I comandi a volte devono essere accompagnati da alcuni parametri che ne precisano il significato.
Alcuni comandi si servono di operatori, che possono essere di tipo aritmetico, relazionale e logico.

I comandi Basic agiscono su dati che possono essere:

.. 1) costanti;
.. 2) variabili;
.. 3) numeri di linea;
.. 4) indirizzi di memoria.

Riassumendo, le linee di programma comprendono i seguenti elementi:

.. numero di linea;
.. parole chiave;
.. costanti;
.. variabili;
.. operatori;
.. caratteri separatori.

I caratteri separatori ammessi sono:

.. lo spazio;
.. la virgola;
.. il punto e virgola;
.. i due punti.

Possiamo dividere i comandi nelle seguenti categorie:

.. 1) Ingresso/Uscita dati
.. 2) Ingresso/Uscita programmi
.. 3) Trattamento interno dati
.. 4) Controllo
.. 5) Servizio
.. 6) Funzioni.

Nel gruppo 6) delle funzioni possiamo fare le seguenti ulteriori suddivisioni:

.. 6.1) Funzioni matematiche
.. 6.2) Funzioni stringa
.. 6.3) Funzioni conversione formato
.. 6.4) Funzioni controllo

.. 6.5) Funzioni lettura memoria
.. 6.6) Funzioni stampa.

Nei prossimi paragrafi, dopo aver parlato delle variabili, delle costanti e degli operatori, ci soffermiamo sulle caratteristiche di ogni istruzione del linguaggio.

L'alfabeto del linguaggio è composto dai seguenti caratteri:
.. le 26 lettere dell'alfabeto
.. le 10 cifre decimali
.. lo spazio
.. 20 caratteri speciali

riportati, salvo lo spazio, nella tabellina che segue.

```
CARATTERI DEL BASIC

A B C D E F G H I J
K L M N O P Q R S T
U V W X Y Z
0 1 2 3 4 5 6 7 8 9
" # $ % ' ( ) * + , -
/ : ; < = > ? @ ↑
```

La tabellina su esposta è stata ottenuta su stampante dal programma che segue; potrai tornare ad esaminarlo quando avrai imparato il linguaggio Basic.

```
1 REM TABCARATTERI
10 OPEN4,4:CMD4
15 PRINTCHR$(14)"CARATTERI DEL BASIC"
20 PRINT:PRINT
30 K1=65:K2=74:GOSUB100:PRINT
33 K1=75:K2=84:GOSUB100:PRINT
35 K1=85:K2=90:GOSUB100:PRINT
40 K1=48:K2=57:GOSUB100:PRINT
50 K1=34:K2=37:GOSUB100
55 K1=39:K2=45:GOSUB100:PRINT
57 PRINTCHR$(47);" ";
60 K1=58:K2=64:GOSUB100
65 PRINTCHR$(94)CHR$(15)
70 PRINT#4:CLOSE4:STOP
100 FORK=K1TOK2
105 PRINTCHR$(K);" ";:NEXTK:RETURN
```

All'interno delle stringhe sono ammessi tutti i caratteri stampabili e alcuni caratteri di controllo.

2.2 COSTANTI E VARIABILI

Le COSTANTI che possono comparire nelle frasi Basic possono essere:

.. Numeri interi, compresi tra -32768 e + 32767.

.. Numeri reali, che vengono stampati in forma decimale fino a 9 cifre, per passare poi alla notazione esponenziale (floating point), vedi il Capitolo 4 del secondo Volume per gli ordini di grandezza.

.. Stringhe di caratteri alfanumerici, delimitate dal carattere virgolette (apici), che non può far parte della stringa. La lunghezza della costante stringa che fa parte della linea di programma deve essere tale da poter essere contenuta negli 80 caratteri consentiti per la linea. Le stringhe non possono comunque superare la lunghezza di 255 caratteri.
Le costanti sono incorporate nelle linee del programma e non sono richiamabili in altri punti del programma. Questo significa che l'uso di costanti può portare a spreco di memoria. Per esempio, se in una istruzione si usa la costante numerica 1984, e in altre 5 istruzioni viene usata ancora la stessa costante numerica 1984, ogni volta vengono usati 4 byte per contenere i 4 caratteri: 1, 9, 8, 4. In tale caso è meglio assegnare un nome simbolico alla costante 1984, che entra così a far parte della categoria delle variabili, e nelle 6 istruzioni richiamare il numero con il nome simbolico della variabile.

Nella memoria del calcolatore sono contenute due costanti speciali; una è richiamabile con il tasto che reca sul fronte il simbolo π e questo risulta il suo nome, l'altra si ottiene usando la funzione EXP(1) ed è il numero irrazionale "e", base dei logaritmi naturali.

Le variabili possono contenere gli stessi tre tipi di dati visti per le costanti; si hanno:

.. Variabili numeriche intere;
.. Variabili numeriche reali;
.. Variabili stringa.

Inoltre si hanno le variabili logiche che non vengono individuate da un nome specifico, il cui valore è assegnabile a variabili numeriche o può fungere da costante numerica nel calcolo di una espressione.

I tre tipi di variabili sono facilmente distinguibili in base al nome che viene loro

assegnato. I NOMI DELLE VARIABILI sono scelti dal programmatore, rispettando le seguenti regole:

.. il nome può essere lungo a piacere, ma vengono usati solo i primi due caratteri per distinguere tra loro le variabili, per questo due variabili diverse non devono mai cominciare con gli stessi due caratteri;

.. il nome di una variabile non deve contenere alcuna sequenza di caratteri che sia una parola chiave del Basic;

.. il nome di una variabile intera viene completato da un suffisso, il carattere %;

.. il nome di una variabile stringa viene completato da un suffisso, il carattere $;

.. il nome di una variabile reale termina senza suffisso;

.. i caratteri validi per formare il nome di una variabile sono le 26 lettere dell'alfabeto e le 10 cifre decimali, però il primo carattere del nome deve essere una lettera.

Esempi di nomi di variabili sono:

.. per numeri interi:

A1% AC% B% PLUTO% (che viene riconosciuto come PL%);

.. per numeri reali:

A1 BD CASA (che viene riconosciuto come CA);

.. per stringhe:

A1$ C$ W$ BELLO$ (che viene riconosciuto come BE$).

Non si può scrivere:

.. TOPO%, infatti viene riconosciuto come TO% e TO è una parola chiave;

.. GOLF, infatti contiene la parola chiave GO;

.. TIPO e TICCHIO vengono riconosciute come la variabile TI.

Si può usare: TA% insieme a TA e insieme a TA$, si tratta infatti di tre variabili diverse.

Qualora in un programma venga richiamata una variabile, alla quale non è ancora stato assegnato un valore, il sistema considera il valore zero per la variabile numerica e il valore di stringa nulla per la variabile stringa. La stringa nulla è una stringa di lunghezza zero, cioè formata da nessun carattere, cosa molto diversa dalla stringa formata da un carattere spazio.

Le due stringhe: A$=" " (uno spazio)
B$="" (stringa nulla)

sono diverse, la prima ha lunghezza 1 e contiene il carattere "spazio", la seconda ha lunghezza 0 e non contiene caratteri.

Una stringa può contenere qualunque carattere salvo le virgolette; infatti le virgolette delimitano le stringhe. Per poter fare entrare le virgolette in una stringa bisogna passarle con la funzione CHR$ usando come argomento il codice ASCII 34.

Le variabili vengono riempite, cioè contengono dei dati, per effetto di una delle seguenti operazioni:

.. assegnazione di un dato. Si ottiene scrivendo il nome della variabile seguito dal carattere "=" (uguale) e dal dato; esempi:

A$ ="pippo" A%=897 A=4567.98

In questa istruzione di assegnazione si può usare la parola chiave LET prima del nome della variabile.

.. lettura di un dato. Si ottiene scrivendo il nome della variabile dopo la parola chiave che indica un'operazione di lettura. L'operazione di lettura può avvenire dall'esterno con la parola chiave INPUT o dall'interno del programma con la parola chiave READ.

Le variabili considerate fino ad ora sono VARIABILI SINGOLE, cioè un nome corrisponde a un dato.

Esistono le VARIABILI DI GRUPPO; esse vengono abitualmente chiamate VARIABILI CON INDICE. Un solo nome definisce un gruppo di variabili; all'interno del gruppo si usano degli indici numerici per distinguere le singole variabili. Esse vengono spesso chiamate ARRAY o MATRICI.
In molti casi si trattano dati che appartengono, per una qualche ragione, ad un gruppo. Una variabile con indice può avere una o più dimensioni, cioè per individuarla può essere sufficiente un solo indice, o possono servirne più di uno. In questa implementazione del Basic non esiste un limite teorico al numero delle dimensioni

di una variabile con indice. Per numero delle dimensioni intendiamo il numero di indici assegnabili.
E' immediato paragonare una variabile con un solo indice ad una lista di variabili. Analogamente, pensando a una variabile con due indici, viene subito in mente una tabella, con un certo numero di righe e di colonne, nella quale ogni variabile rappresenta un nodo, cioè il punto di incontro tra una riga e una colonna.
Continuando, una variabile con tre indici può far venire in mente un parallelepipedo; ogni variabile si trova su una riga e su una colonna di un piano, cioè a una certa quota.

Si può continuare, ma la geometria elementare non ci viene più facilmente in aiuto. Non devi confondere il numero delle dimensioni con l'ordine di grandezza di ogni dimensione.

Una lista di 100 variabili può esser pensata come una variabile a una dimensione formata da 100 elementi. Una variabile relativa a 20 argomenti, ognuno formato da 3 dati, viene immaginata come una tabella formata da 20 righe di 3 colonne ciascuna; sono in tutto 60 elementi, la dimensione riga varia da 1 a 20, mentre la dimensione colonna varia da 1 a 3.
Il numero degli elementi in ogni dimensione può raggiungere il valore massimo per gli interi positivi, cioè 32767. Il numero delle dimensioni può raggiungere 255. Ma attenzione, 255x32767 supera il numero dei byte della memoria del nostro calcolatore!

Per la definizione delle variabili con indice esiste una istruzione apposita: DIM (DIMension). Questa istruzione deve essere obbligatoriamente usata per poter trattare variabili con indice per le quali almeno una dimensione superi gli 11 elementi. Qualora si stia al disotto di questo valore, non è necessario usare la istruzione DIM.

Le variabili con indice possono essere dei tre tipi già trattati: intere, reali e stringhe. Per la formazione dei nomi valgono le regole già viste; il nome delle variabili viene seguito da una coppia di parentesi, all'interno delle quali si scrivono gli indici, separati da virgole, se sono più di uno.
Dopo aver eseguito l'operazione di dimensionamento, il sistema assegna il valore 0 alle variabili numeriche e il valore stringa nulla alle variabili stringa.

Esempi:

DIM A(24) crea la variabile con indice A, con un solo indice (tra le parentesi compare un solo numero), formata da 25 elementi numerici reali e assegna il valore zero ad ogni elemento. Il primo elemento è A(0) e l'ultimo è A(24). Gli indici partono dal valore zero.

DIM BR$(15) crea la variabile con indice BR$, formata da 16 stringhe, tutte al valore iniziale di stringa nulla.

Se nel programma si cita la variabile B%(5), che non è stata dimensionata, il sistema fa un dimensionamento implicito, come se si fosse scritto DIM B%(10), cioè crea una variabile con indice formata da 11 elementi interi con valore iniziale uguale a zero.

Nel Capitolo 8 descriviamo in dettaglio l'occupazione di memoria per ogni tipo di variabile.

Le VARIABILI LOGICHE sono variabili che possono assumere due soli valori: VERO e FALSO. In realtà esse possono essere considerate variabili numeriche. Nel prossimo paragrafo trattiamo ampiamente l'argomento.

2.3 OPERATORI ARITMETICI, RELAZIONALI E LOGICI

Il Basic accetta ESPRESSIONI formate da COSTANTI e VARIABILI collegate tra loro da OPERATORI ARITMETICI e/o RELAZIONALI e/o LOGICI. Nella spiegazione dettagliata delle istruzioni del linguaggio, nel Paragrafo 2.4, abbiamo sempre messo in rilievo quando si possono usare espressioni.

Gli OPERATORI ARITMETICI accettati sono:

Operatori	*Significato*
+	somma
—	sottrazione
*	moltiplicazione
/	divisione
↑	elevamento a potenza
=	uguale a
(	aperta parentesi
)	chiusa parentesi

con i normali significati dell'aritmetica.

Gli OPERATORI RELAZIONALI accettati sono:

Operatori	*Significato*
<	minore di
=	uguale a
>	maggiore di
<=	minore di o uguale a
>=	maggiore di o uguale a
<>	non uguale a

e servono a stabilire una relazione sia in senso artitmetico che non aritmetico.

Gli OPERATORI LOGICI accettati sono:

Operatori	*Significato*
AND	e (l'uno e l'altro)
OR	o (uno o l'altro)
NOT	no (negazione)

e possono collegare tra loro sia espressioni aritmetiche che relazionali.

Le espressioni vengono elaborate partendo da sinistra e procedendo verso destra, rispettando le regole di precedenza che seguono (riportate dalla più alta alla più bassa), ma tenendo presente che le espressioni racchiuse in parentesi hanno a loro volta la precedenza.

REGOLE DI PRECEDENZA

↑	elevamento a potenza
-	negazione (segno meno)
* /	moltiplicazione e divisione
+ -	somma e sottrazione
> = <	relazione
NOT	NO logico
AND	AND logico
OR	OR logico

Nell'elaborazione delle espressione aritmetiche vengono applicate integralmente le regole dell'algebra; devi fare attenzione al tipo delle variabili aritmetiche che sono coinvolte e ricordare queste regole:

.. ogni operazione avviene tra due operandi;
.. sia che i due operandi siano interi, sia che siano uno intero e l'altro reale, sia che siano tutti e due reali il risultato è reale;
.. alla fine del calcolo il valore viene assegnato alla variabile che sta a sinistra dell'uguale rispettandone il tipo.

Per poter ottenere risultati interi, senza doverli necessariamente assegnare ad una variabile di tipo intero, si può usare la funzione INT, che fornisce la parte intera di un risultato.

L'unica operazione aritmetica eseguibile tra stringhe è la somma (+), nel senso che sommare due stringhe significa scriverle una di seguito all'altra, ottenendo una stringa più lunga (massimo 255 caratteri).

Gli operatori relazionali si usano per confrontare tra loro due operandi; questo confronto ha il normale significato aritmetico per due operandi numerici, mentre ha un significato basato sull'ordinamento per gli operandi di tipo stringa. Le stringhe sono formate da caratteri ASCII; ogni carattere corrisponde a un codice numerico. Quando si confrontano tra loro due stringhe, esse sono confrontate carattere per carattere in base al valore numerico del codice. In tale confronto viene rispettato l'ordine alfabetico normale. Se si confronta una stringa, con una più lunga, ma che ha i primi caratteri uguali, la più lunga risulta maggiore.

"ABBA" risulta minore di "ABBAGLIO".

Puoi vedere nel Capitolo 4 del secondo Volume i valori dei codici ASCII nei due set di caratteri.

Nel confronto tra due operandi, di qualunque tipo, nasce una VARIABILE LOGICA, che, da un punto di vista logico ha il significato di VERO o FALSO, ma in realtà ha un valore aritmetico: il numero intero -1 per VERO e il numero intero 0 per FALSO. Per questa ragione in espressioni di tipo aritmetico può essere compresa una espressione relazionale; ad essa, durante il calcolo, viene sostituito il numero intero risultato. Ha quindi senso scrivere:

A%=C$>B$ A% contiene -1 se C$ è maggiore di B$
A% contiene 0 se C$=B$ o se C$<B$.

Si possono operare confronti tra numeri interi e numeri reali; per esempio:

A=1237 e A%=1237 se confrontati risultano uguali.

Devi fare particolare attenzione ai confronti tra numeri che risultano da calcoli complessi, dato che, anche se quando stampi due numeri essi sono uguali, in memoria possono essere leggermente diversi; sono scherzi dovuti al calcolo binario. Senza andare in calcoli troppo complicati, considera il programma CALCOLI che segue, e i suoi rusultati; ti aspetteresti che A e B siano uguali, mentre non è così.

```
1 REM CALCOLI
10 OPEN4,4:CMD4
20 A=(3↑(-10))
30 B=(1/3)↑10
40 PRINT"PRIMO CALCOLO: (3↑(-10)) =";A
50 PRINT"SECONDO CALCOLO: (1/3)↑10=";B
60 PRINT"DIFFERENZA CALCOLATA:     =";A-B
70 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA CALCOLI

```
PRIMO CALCOLO: (3↑(-10)) = 1.69350878E-05
SECONDO CALCOLO: (1/3)↑10= 1.69350879E-05
DIFFERENZA CALCOLATA:     =-1.20792265E-13
```

Gli OPERATORI LOGICI (detti anche BOOLEANI) servono per collegare tra loro due o più espressioni di qualunque tipo. Essi possono essere usati per arricchire il significato di espressioni di tipo relazionale, o per considerare risultati aritmetici diversi in relazione tra loro. L'applicazione di questi operatori produce un risultato logico di VERO o FALSO espresso in modo aritmetico; se il numero assegnato alla variabile logica è 0 (zero) il risultato viene considerato falso in una analisi di condizione, mentre se esso è diverso da zero (non sempre risulta -1) viene considerato vero.

Gli operatori logici lavorano confrontando i bit corrispondenti con i seguenti risultati:

Oper. logica	Risult. aritmetico	Risult. relazionale
1 AND 1	1	VERO
1 AND 0	0	FALSO
0 AND 1	0	FALSO
0 AND 0	0	FALSO
1 OR 1	1	VERO
1 OR 0	1	VERO
0 OR 1	1	VERO
0 OR 0	0	FALSO
NOT 1	0	FALSO
NOT 0	1	VERO

Nel programma OPERATORI LOGICI che segue si stampano 16 esempi per capire come lavorano gli operatori logici.

```
1 REM OPERATORI LOGICI
3 OPEN4,4:CMD4
5 PRINT"ESEMPI USO OPERATORI LOGICI":PRINT
10 A=27:B=90:C=45:D=4
20 PRINT" 1) ";"A<B AND D<C    ";A<BANDD<C
25 X=A<BANDD<C
30 PRINT" 2) ";"X=A<B AND D<C =";X
```

```
40 PRINT" 3) ";"A>B AND D<C   ";A>BANDD<C
45 Y=A>BANDD<C
50 PRINT" 4) ";"Y=A>BANDD<C =";Y
60 Z=AANDB
65 PRINT" 5) ";"Z=A AND B   ";Z
70 PRINT" 6) ";"A<B OR D<C   ";A<BORD<C
73 X=A<BORD<C
75 PRINT" 7) ";"X=A<B OR D<C =";X
80 PRINT" 8) ";"A>B OR D<C : ";A>BORD<C
83 Y=A>BORD<C
85 PRINT" 9) ";"Y=A>BORD<C =";Y
87 Z=AORB
89 PRINT"10) ";"Z=A OR B : ";Z
90 PRINT"11) ";"NOT B : ";NOT B
93 IFNOTA>BTHENPRINT"12) ";"NOT A>B; VERO: A<B"
95 PRINT"13) ";"NOT A > B : ";NOTA>B
97 IFNOTB<ATHENPRINT"14) ";"NOT B<A; VERO: A<B"
99 PRINT"15) ";"NOT A < B : ";NOTA<B
100 Z=NOTA
105 PRINT"16) ";"A=";A,"NOT A=";Z
110 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA OPERATORI LOGICI

```
ESEMPI USO OPERATORI LOGICI

 1) A<B AND D<C : -1
 2) X=A<B AND D<C =-1
 3) A>B AND D<C :  0
 4) Y=A>BANDD<C = 0
 5) Z=A AND B :  26
 6) A<B OR D<C : -1
 7) X=A<B OR D<C =-1
 8) A>B OR D<C : -1
 9) Y=A>BORD<C =-1
10) Z=A OR B :  91
11) NOT B : -91
12) NOT A>B; VERO: A<B
13) NOT A > B : -1
14) NOT B<A; VERO: A<B
15) NOT A < B :  0
16) A= 27            NOT A=-28
```

All'inizio del programma vengono assegnati quattro valori numerici alle quattro variabili: A, B, C, e D. Dopo si mandano alla stampante sedici risultati che commentiamo qui.

1) A<B AND D<C fornisce il risultato aritmetico -1 nel senso di relazione VERA, infatti : 27<90 e 4<45.

2) Assegnando alla variabile X il risultato della precedente relazione composta si ottiene -1, dato che la relazione è VERA.

3) A>B AND D<C fornisce il risultato aritmetico 0 nel senso di relazione FALSA.

4) Assegnando alla variabile Y il risultato della precedente relazione composta si ottiene 0, dato che la relazione è FALSA.

5) Z = A AND B dà il risultato 26 perchè, considerando i valori binari dei numeri si ha:

A=27	in binario	A=00011011
B=90	in binario	B=01011010
A AND B	in binario	Z=00011010 Z=26

infatti per effetto della operazione AND si ha un bit 1 dove tutti e due gli operandi hanno 1.

6) A<B OR D<C fornisce il risultato aritmetico -1 nel senso di relazione VERA. In particolare viene eseguito OR di due relazioni vere, per dare risultato -1 basta che sia vera una delle due.

7) Sempre per la stessa ragione del punto 6) X riceve il valore -1.

8) In questo caso si ha OR di due relazioni, una vera e una falsa, e il risultato è VERO.

9) Vale quanto detto in 8).

10) Si opera OR tra A e B, con risultato 91; considerando i valori binari si ha:

A=27	in binario	A=00011011
B=90	in binario	B=01011010
A OR B	in binario	Z=01011011 Z=91

infatti si ottiene un bit 1 dove anche uno solo dei due operandi ha 1.

11) NOT B =—91, infatti B=90, consideriamo i valori binari:

    B=90 in binario B=01011010 e operando con NOT su ogni bit si ottiene:

    NOT B=10100101 che interpretato come numero, dato che inizia con un bit 1, dà il valore negativo —91.

12) NOT A>B risulta VERO, infatti la relazione A>B è FALSA; l'operatore NOT viene applicato alla relazione. Se si prova, per gli stessi valori di A e B:

    PRINT (NOTA)>B si ottiene 0, infatti in questo caso NOT viene applicato ad A e NOT A=—28 risulta <90, e quindi la relazione è FALSA.

13) Vale quanto detto all'inizio del punto 12).

14) L'operatore NOT si applica ad una relazione falsa e quindi dà un risultato VERO.

15) L'operatore NOT si applica ad una relazione vera e quindi dà un risultato FALSO, uguale 0 aritmeticamente.

16) Per quanto visto già al punto 11), essendo A=27, NOT A=—28. Puoi provare a rifare i calcoli in binario.

Ci rendiamo conto che i programmi esempio riportati compaiono troppo presto; non sono infatti ancora state spiegate tutte le istruzioni. Se fai fatica a seguire i programmi, potrai riprenderli in esame più tardi.

Seguono ulteriori spiegazioni sugli operatori logici, corredate da esempi.

AND Tipo: Operatore logico

Formato: espressione AND espressione

Abbiamo già messo in evidenza, sia l'aspetto aritmetico, che quello di controllo tra due relazioni, di questo operatore logico.

Puoi provare il programma che segue per diversi valori dei due numeri A e B e riflettere sui risultati.

```
1 REM ES2 PROVA AND
5 A$="A>B AND B<>0"
10 INPUT"PRIMO: ";A
20 INPUT"SECONDO: ";B
30 PRINT"A AND B";AANDB
40 PRINT"A AND NOT B";AANDNOTB
50 PRINT"NOT A AND B";NOTAANDB
60 IFA>BANDB<>0THENPRINTA$:GOTO80
70 PRINT"NON VERIFICATO"
80 STOP
```

Dato che l'operatore agisce su numeri compresi tra -32768 e +32767, se dai per A e/o B valori fuori dai limiti, avrai il messaggio di errore: ?ILLEGAL QUANTITY.

NOT Tipo: Operatore logico

Formato: NOT espressione

dove espressione può essere una qualunque espressione numerica.

Si tratta di un operatore logico che può essere usato in modo aritmetico o logico. Se l'espressione numerica dà un risultato reale, l'operatore lavora solo sulla parte intera, che deve essere compresa tra —32768 e +32767. In caso contrario si ha il messaggio: ?ILLEGAL QUANTITY. Quando lavora in senso aritmetico su un numero N, il numero NOT N è: -(N+1).

Segue un programma che ti consente di riflettere sul significato di questo operatore.

```
1 REM ES36 PROVA NOT
5 R$="NOT(N1<N2): "
10 INPUT"NUMERO: ";N
20 PRINT"N=";N;"NOT N=";NOTN
30 INPUT"SCRIVI 2 NUMERI: ";N1,N2
40 PRINT"I DUE NUMERI SONO: ";N1;" , ";N2
50 IFN1>N2THENPRINT"N1>N2":GOTO60
55 PRINT"N1<=N2"
60 IFNOTN1<N2THENPRINT"NOTN1<N2: ";NOTN1<N2
63 PRINT"NOTN1: ";NOTN1;"N2: "N2
65 IFNOT(N1<N2)THENPRINTR$;NOT(N1<N2)
67 PRINT"N1<N2: ";N1<N2
70 STOP
```

OR Tipo: Operatore logico

Formato: espressione OR espressione

dove espressione può essere una relazione logica o una espressione numerica.

Questo operatore logico può essere usato per correlare tra loro due relazioni e quindi può intervenire in una scelta tra vero e falso. Nell'uso numerico valgono le regole viste.

L'operatore lavora su numeri interi compresi tra -32768 e +32767, con operandi fuori da questi limiti si ha segnalazione di errore.

Segue un programma che esemplifica quanto detto.

```
1 REM ES39 PROVA OR
10 INPUT"SCRIVI 2 NUMERI";N1,N2
20 PRINT"N1=";N1;" N2=";N2
30 PRINT"N1 OR N2 = ";N1ORN2
40 IFN1<N2ORN1<300THENPRINT"N1<N2 OR N1<300"
50 IFN1<0ORN2<0THENPRINT"ALMENO UNO NEGATIVO"
```

2.4 ISTRUZIONI

In questo paragrafo elenchiamo le istruzioni del linguaggio in ordine alfabetico, onde consentirne una facile ricerca in caso di dubbi.

Ricordiamo che le linee di un programma, che deve essere eseguito in modo differito, si scrivono numerandole progressivamente. E' buona regola non usare numeri consecutivi; questo facilita eventuali inserzioni nella fase di prova del programma.
Una linea di programma può contenere più istruzioni; in tale caso devi usare come carattere separatore tra le istruzioni il carattere ":" (due punti).
Tutte le istruzioni, salvo quella di assegnazione, iniziano con una parola chiave. Per l'istruzione di assegnazione la parola chiave LET è facoltativa.
Una linea di programma non può superare gli 80 caratteri, 2 righe del video. In realtà si riescono a superare gli 80 caratteri, che costituiscono un limite nella fase della scrittura del programma, usando le abbreviazioni consentite per le parole chiave. Questo però presenta un rischio; infatti quando si chiede di listare sul video il programma, sono listate anche le istruzioni che, riportando in modo esteso le parole chiave, superano gli 80 caratteri, però, se si apporta qualche correzione alla linea, quando si preme RETURN, la linea viene troncata. Per evitare questo

inconveniente si dovrebbe tornare a contrarre le parole chiave per far entrare di nuovo la linea di programma in 80 caratteri. Questo modo di procedere risulta un poco macchinoso. Ti consigliamo pertanto di non scrivere linee di programma troppo lunghe.

Alcune istruzioni non possono essere usate in modo immediato, mentre per la maggior parte questo è possibile. Nel seguito segnaliamo esplicitamente le istruzioni che non possono essere usate in modo immediato e quelle che non ha senso usare in modo differito, anche se è consentito farlo.

Nel Paragrafo 2.1 abbiamo definito sei categorie per le istruzioni Basic; ora elenchiamo le istruzioni che appartengono ad ogni gruppo.

1) Ingresso e Uscita dati, abbreviato in Operazioni I/O, da Input/Output:

CLOSE	INPUT#
CMD	OPEN
GET	PRINT
GET#	PRINT#
INPUT	

2) Operazioni I/O programmi:

LOAD	VERIFY
SAVE	

3) Trattamento interno dati:

DATA	POKE
DIM	READ
LET	RESTORE

4) Controllo:

CONT	ON
END	RETURN
FOR.. TO.. STEP	RUN
GOSUB	STOP
GOTO	SYS
IF.. THEN	WAIT
NEXT	

5) Servizio:

CLR	NEW
DEF FN	REM
LIST	

6) Funzioni così ripartite:

6.1) Matematiche
ABS
ATN
COS
EXP
FN
INT
LOG
RND
SGN
SIN
SQR
TAN

6.2) Stringa
CHR$
LEFT$
MID$
RIGHT$
STR$

6.3) Conversione
ASC
VAL

6.4) Controllo
USR

6.5) Lett. memoria
FRE
LEN
PEEK
POS
STATUS
TIME
TIME$

6.6) Stampa
SPC
TAB

Prima di passare alla descrizione delle istruzioni preferiamo giustificare i raggruppamenti.

Abbiamo definito il gruppo 1 per le istruzioni che riguardano l'entrata e l'uscita dei dati rispetto al calcolatore. Il gruppo 2 è stato invece definito per l'entrata e l'uscita dei programmi. Il gruppo 3 riguarda quelle operazioni che modificano i contenuti della memoria, vuoi per semplici trasferimenti che per calcoli eseguiti. Nel gruppo 4 rientrano tutte quelle istruzioni che controllano lo svolgimento del programma. Nel gruppo 5 abbiamo lasciato poche istruzioni di servizio. Nel gruppo 6 abbiamo raggruppato tutte le funzioni, procedendo poi ad altre 6 suddivisioni in base al tipo delle funzioni stesse. Per esempio in 6.4) entra la funzione USR che manda in esecuzione un programma in linguaggio macchina e quindi agisce come la SYS del gruppo 4. La CMD, che abbiamo messo nel gruppo 1, in realtà potrebbe anche essere messa altrove, per esempio nel gruppo 5, tra le istruzioni di servizio. Qualcuno potrebbe ritenere più appropriato porre OPEN e CLOSE ancora nel gruppo 5.

Si potrebbe discutere a lungo su queste suddivisioni; anche noi non siamo sicuri di avere fatto buone scelte. Ti invitiamo pertanto ad entrare nella discussione, quando sarai diventato un esperto del linguaggio Basic. Una proposta potrebbe essere che, essendo il Basic un linguaggio di tipo interpretativo, qualunque istruzione venga considerata una funzione svolta servendosi di uno o più sottoprogrammi.

Nel seguito, nella descrizione delle istruzioni, si usano le seguenti convenzioni:

.. per ogni istruzione si scrivono la o le parole chiave, in stampatello;
.. si cita il gruppo di appartenenza;
.. si espone il formato di scrittura;
.. gli operandi e i parametri sono scritti in caratteri minuscoli;
.. le parti opzionali sono racchiuse in parentesi quadre;
.. i puntini significano ripetizione;
.. per i parametri e gli operandi usiamo alcune abbreviazioni:

lfn	numero-logico-file
var	variabile
fn	nome-file
dn	numero-logico-periferica (device number)
sa	indirizzo-secondario
ft	tipo-file
md	modo

.. espressione significa una qualunque espressione, al limite una sola variabile o costante.

Riportiamo la tabellina delle dn relative alle più comuni periferiche.

PERIF.	DN	SA
Cassette	1	0 = Input 1 = Output 2 = Output con EOT
Video	3	
Stampante	4 o 5	0 = Set maiuscolo/grafico 7 = Set maiuscolo/minuscolo
Disco	8/11	0/1 = trasferimento programmi 2/14 = canali dati 15 = canale comandi

ABS Tipo: Funzione matematica

Formato: ABS (espressione)

dove "espressione" deve dar luogo ad un valore numerico.

La funzione ABS fornisce il valore assoluto dell'argomento, cioè l'argomento stesso, se esso è positivo, l'argomento moltiplicato per -1, se esso è negativo. Puoi provare il programma che segue; ti viene chiesto un numero e sono stampati sul video il numero e il suo valore assoluto.

```
1 REM ES1 PROVA ABS
10 INPUT"SCRIVI UN NUMERO ";X
15 A$=" VALORE ASSOLUTO DI X="
20 PRINT"X=";X;A$;ABS(X)
```

ASC Tipo: Funzione conversione formato

Formato: ASC (stringa)

dove stringa può essere sia una stringa delimitata dalle virgolette, cioè una costante, che una variabile stringa.

Il risultato fornito da questa funzione è un numero compreso tra 0 e 255, tale numero è il codice ASCII del primo carattere della stringa argomento. Si può avere il messaggio di errore: ?ILLEGAL QUANTITY, se la stringa argomento è la stringa nulla.

Puoi provare il programma che segue, rispondendo alla richiesta con diverse stringhe; puoi anche rispondere con la stringa nulla; infatti alla linea 20 abbiamo usato un accorgimento aggiungendo alla stringa A$ la stringa di un carattere CHR$(0). Puoi verificare sulla tabella dei codici ASCII il codice numerico stampato.

```
1 REM ES3 PROVA ASC
10 INPUT A$
20 PRINTASC(A$+CHR$(0))
30 STOP
```

ATN Tipo: Funzione matematica

Formato: ATN (numero)

dove numero può anche essere una variabile numerica o una espressione numerica.

Il risultato è l'angolo, espresso in radianti, che ha come tangente l'argomento fornito. Puoi provare il programma che segue per diversi valori dell'argomento.

```
1 REM ES4 PROVA ATN
2 INPUT"ARGOMENTO: ";A
3 OPEN4,4:CMD4
4 A$="ATN(":C$="GRADI: "
5 B$=")=":D$="RAD.: "
6 PRINTD$A$;"1";B$;ATN(1)
7 PRINTC$A$;"1";B$;ATN(1)*180/π
8 PRINTD$A$;"0";B$;ATN(0)
9 PRINTC$A$;"0";B$;ATN(0)*180/π
20 PRINTD$A$;A;B$;ATN(A)
25 PRINTC$A$;A;B$;ATN(A)*180/π
30 PRINT#4:CLOSE4:STOP
```

Ecco alcuni risultati del programma:

```
RAD.: ATN(1)= .785398163
GRADI: ATN(1)= 45
RAD.: ATN(0)= 0
GRADI: ATN(0)= 0
RAD.: ATN( 38 )= 1.54448661
GRADI: ATN( 38 )= 88.4925643
```

Come vedi alle linee 6, 8 e 25 si trasforma il risultato in gradi. Il risultato fornito dalla funzione è sempre compreso tra $-\pi/2$ (-1.57079633) e $+\pi/2$ (+ 1.57079633).

CHR$ Tipo: Funzione stringa

Formato: CHR$ (numero)

dove numero può anche essere una variabile o un'espressione numerica, ma l'argomento deve essere compreso tra 0 e 255 (valori possibili per 1 byte). La funzione fornisce una stringa di un carattere, il carattere che corrisponde al codice. Se il codice si riferisce a un carattere non stampabile la stringa prodotta ha lunghezza 1, ma stampandola non compare alcunchè. La funzione contraria di CHR$ e ASC.

Puoi provare il programma che segue e vedrai confermato quanto abbiamo detto.

```
1 REM ES5 PROVA CHR$
10 PRINT"CHR$(0)=";CHR$(0)
20 PRINT"CHR$(255)=";CHR$(255)
30 PRINT"CHR$(127.67)=";CHR$(127.67)
40 PRINT"CHR$(127)=";CHR$(127)
43 A=56:A$=CHR$(A):B=ASC(A$)
45 PRINT"CHR$(A)=";CHR$(A)
47 PRINT"ASC(A$)=";ASC(A$)
50 INPUT"CODICE ASCII ";A
60 PRINT"CHR$(";A;")=";CHR$(A)
70 STOP
```

RISULTATI PROGRAMMA ES5

```
CHR$(0)=
CHR$(255)=π
CHR$(127.67)=◥
CHR$(127)=◥
CHR$(A)=8
ASC(A$)= 56
CODICE ASCII CHR$( 48 )=0
```

Se l'argomento di CHR$ è >255 si ha il messaggio di errore: ?ILLEGAL QUANTITY, mentre se esso è un numero con decimali, ma nei limiti consentiti, viene presa la parte intera e non si ha segnalazione di errore.

CLOSE Tipo: Operazione di I/O dati

Formato: CLOSE lfn

dove lfn, che può essere una costante o una variabile, è il numero logico del file usato nella operazione di OPEN che ha consentito di lavorare con quel file usando un certo canale. Fai attenzione che nella CLOSE si usa il numero logico del file e non il numero del canale. Pensiamo sia necessario puntualizzare questo fatto, dato che spesso si ha l'abitudine di usare gli stessi numeri per canale e numero logico file. L'operazione CLOSE è molto importante perchè quando si lavora con unità di uscita come cassette o floppy, che usano un BUFFER per ammucchiare i dati da scrivere, è proprio essa che va a scrivere il contenuto dell'ultimo blocco e registra la chiusura del file. Un file non chiuso, non può poi essere riletto. Questa operazione va comunque sempre eseguita a conclusione di una operazione iniziata con una OPEN; infatti nella memoria di lavoro del calcolatore, per ogni OPEN viene memorizzato un insieme di informazioni che continua a sussistere fino alla CLOSE corrispondente ed è dannoso tenere occupata la relativa tabella con informazioni

che non servono più.
Nel programma che segue con un ciclo si chiudono tutti i file con numero-logico da K1 a K2.

```
1 REM ES6 PROVA CLOSE
10 REM CHIUSURA DI TUTTI I FILE
20 FORK=K1TOK2
30 CLOSEK
40 NEXTK
```

CLR Tipo: Servizio

Formato: CLR

Questa istruzione PULISCE tutte le variabili del programma, cioè pone zero in quelle numeriche e stringa-nulla in quelle stringa. Inoltre annulla tutte le memorizzazioni relative ad operazioni in corso come: GOSUB, cicli FOR...NEXT, funzioni definite dall'utente, OPEN di file. Quest'ultimo effetto può essere pericoloso, perchè eventuali file ancora aperti non vengono chiusi in modo corretto; per il sistema disco, ad esempio, i file sono ancora aperti, mentre per il calcolatore non lo sono più.
Segue un programma dimostrativo e sono riportati i risultati come appaiono sul video.

```
1 REM ES7 PROVA CLR
10 A=34.15:A%=89:A$="PROVA CLR"
20 PRINT"STAMPA VARIABILI":GOSUB50
30 CLR
40 PRINT"STAMPA DOPO CLR":GOSUB50
45 STOP
50 PRINT"A=";A;" A%=";A%;" A$=";A$
55 RETURN
```

RISULTATI PROGRAMMA PRECEDENTE

```
STAMPA VARIABILI
A= 34.15  A%= 89  A$=PROVA CLR

STAMPA DOPO CLR
A= 0  A%= 0  A$=


BREAK IN 45
```

CMD Tipo: Operazione I/O dati

Formato: CMD lfn [,stringa]

dove lfn è il numero logico di un file che deve essere stato preventivamente aperto con OPEN su una periferica diversa dal video. La stringa che segue è opzionale; se presente viene mandata al file e può essere considerata come titolo.
Questa operazione pone la periferica, aperta con il numero-logico di file specificato, in stato di attesa, sostituendola al video. Le successive operazioni di PRINT e LIST agiscono verso la periferica designata. Per far tornare attivo il video si deve scrivere una istruzione PRINT# lfn, senza operandi; questo corrisponde all'invio di una linea bianca alla periferica, e poi si deve chiudere il file con CLOSE. Se si verifica un errore del tipo: ?SYNTAX ERROR l'uscita dei dati torna verso il video, ma la periferica rimane in stato di attesa e quindi si genera una situazione irregolare; devi quindi usare la procedura sopra descritta anche in questo caso.
Segue un programma esempio che dapprima, linee da 5 a 30, manda un file sequenziale su cassetta, chiamandolo PROVA CASSETTA, e poi apre la stampante come file 4, le invia un messaggio e lista se stesso.

```
1 REM ES8 PROVA CMD
5 REM MANDA FILE SEQUENZIALE SU CASSETTA
10 OPEN1,1,1,"PROVA CASSETTA"
20 FORK=1TO26:PRINTCHR$(64+K):NEXTK
30 PRINT#1:CLOSE1
35 OPEN4,4:CMD4
40 PRINT"SCRITTO FILE SEQ. SU CASSETTA"
45 PRINT"LISTA PROGRAMMA"
50 LIST
```

RISULTATI PROGRAMMA PRECEDENTE

```
SCRITTO FILE SEQ. SU CASSETTA
LISTA PROGRAMMA
```

Dopo l'esecuzione del programma devi in modo immediato scrivere:

PRINT# 4:CLOSE4

per chiudere correttamente, infatti dopo LIST il controllo non torna più al programma in corso.

Fai attenzione al fatto che, se nella lista di stampa sono compresi anche caratteri di controllo per il video, essi possono avere effetti nulli o diversi per altre periferiche.

CONT Tipo: Controllo

Formato: CONT

Si tratta di un comando che non ha senso scrivere come parte di un programma da usare in modo differito. Il suo uso è in modo immediato per far ripartire un programma dal punto dove si è fermato per effetto delle istruzioni STOP o END incontrate o per l'uso estemporaneo del tasto RUN/STOP. Se il programma si ferma per aver incontrato l'istruzione END non compare alcun messaggio. Negli altri due casi compare il messaggio: BREAK IN numero-linea. Scrivendo CONT e premendo RETURN il programma continua.
Devi fare attenzione, quando il programma è fermo, a non svolgere alcuna azione di EDIT; puoi solo chiedere dei LIST totali o parziali, stampare il valore di variabili da controllare, modificare in modo immediato i contenuti di qualche variabile. In tutti questi casi scrivendo CONT il programma procede. Basta che tu, dopo la fermata, chieda un LIST, porti il cursore su una linea e prema RETURN anche senza aver apportato modifiche (operazione di EDIT) perchè al comando CONT venga risposto con il messaggio: CAN'T CONTINUE ERROR.
E' importante imparare ad usare bene CONT, infatti nella fase di messa a punto dei programmi è opportuno piazzare degli STOP in punti strategici (meglio STOP che END per vedere in quale linea si è fermato) e poi andare ad indagare su particolari situazioni delle variabili.
Segue un programma che calcola per approssimazioni successive il valore di π.

```
1 REM ES9 PROVA CONT
10 PI=0:C=1
20 PI=PI+4/C-4/(C+2)
30 PRINT "PI=";PI;"  π="π
40 C=C+4:GOTO20
```

Puoi provare a farlo girare e premere RUN/STOP, poi scrivere in immediato:

PRINT C e poi RETURN

per vedere a quale valore di C sei e a quale punto dell'approssimazione, del valore di π calcolato da te, sei rispetto a quello della macchina. Per rallentare lo scrolling del video puoi tenere premuto il tasto CTRL.
Se provi vedrai che quando C arriva oltre 3370, nel valore calcolato sono giuste le prime 3 cifre decimali.

COS Tipo: Funzione matematica

Formato: COS (numero)

dove numero può anche essere una variabile o un'espressione numerica. L'argomento deve essere la misura di un angolo espresso in radianti.
Segue un esempio, nel quale, se scegli di introdurre l'angolo in gradi, il programma lo trasforma in radianti prima di calcolare il coseno.

```
1 REM ES10 PROVA COS
10 INPUT"GRADI O RADIANTI (G/R) ";R$
20 INPUT"ARGOMENTO: ";A
30 IFR$="G"THENA=A*π/180
40 PRINT"COS(";A;")=";COS(A)
50 STOP
```

DATA Tipo: Trattamento interno dati

Formato: DATA lista-costanti

dove lista-costanti è una serie di costanti anche di tipo diverso, separate dal carattere separatore virgola (,).
Si tratta di una istruzione che non viene propriamente eseguita, per cui non importa dove è sistemata nel programma. Quello che importa, se sono presenti più DATA, è l'ordine nel quale compaiono, cioè la sequenza dei numeri di linea. Tutte le frasi DATA presenti nel programma contribuiscono alla formazione di una lista di dati nell'ordine nel quale sono nel programma. La prima costante della prima frase DATA è il primo dato nella lista, l'ultima costante dell'ultima frase DATA è l'ultimo dato nella lista. Non appena il programma caricato in memoria viene mandato in esecuzione, il sistema predispone un puntatore interno al primo dato della lista generata dalle frasi DATA. Tale puntatore si sposta ogni volta che viene prelevato un dato dalla lista per mezzo della istruzione READ (lettura interna al programma) che lo trasferisce in una variabile. Esiste una istruzione che permette di riposizionare il puntatore all'inizio della lista dei dati, la RESTORE.
I dati della lista possono essere di qualunque tipo, le stringhe devono essere delimitate dalle virgolette (apici) solo se contengono i seguenti caratteri:

:	spazi	lettere digitate insieme a SHIFT
caratteri grafici		caratteri di controllo

devi stare attento quando usi l'istruzione READ per attingere alla lista dei dati e usare nomi di variabili che si accordino con il tipo di variabile puntata al momento.

Usare sempre variabili stringa toglie dagli impicci e non fa rischiare errori.
Questa istruzione non può essere usata in modo immediato.

Segue un esempio di uso di DATA:

```
1 REM ES11 PROVA DATA
10 DATA"PROVA DI DATA","PASSO AI NUMERI"
20 DATA1,2,3,4,5,6,7,8,9,0
30 DATA32767,-32768,127E-14
40 DATAA,B,C,D,E,F,G,H,I,J,K,L,M
50 DATAN,O,P,Q,R,S,T,U,V,W,X,Y,Z
60 DATA"[redacted]"
70 DATAFINE
```

Con le frasi riportate si definisce una lista di 43 costanti.
Nota che sono state scritte tra virgolette le prime due stringhe, infatti contengono degli spazi, la penultima che contiene caratteri di controllo. L'ultima costante, la parola FINE, non necessita di virgolette, ma non sarebbe un errore usarle.

DEF FN Tipo: Servizio

Formato: DEF FN nome(var)=espressione

dove:
DEF FN può essere scritto anche senza spazio DEFFN,
nome è il nome che si vuole assegnare alla funzione che si sta definendo e deve rispettare le regole di formazione dei nomi delle variabili numeriche decimali; la funzione viene richiamata da: FNnome;
var è una variabile che serve come argomento falso per definire matematicamente la funzione; al momento dell'uso la funzione viene richiamata ed eseguita con l'argomento (variabile o costante) che gli viene passato, e che diventa l'argomento vero;
espressione è una espressione di calcolo, nella quale possono essere richiamate anche altre funzioni.

E' consentito l'uso di una sola variabile falsa (dummy); al momento del richiamo della funzione si deve porre tra parentesi l'argomento da usare nel calcolo, che può essere una costante o una variabile. La definizione della funzione, cioè l'esecuzione della linea di programma che la definisce deve avvenire una volta, prima che la funzione possa essere richiamata.
Le funzioni definite dall'utente, dopo la definizione nell'ambito del programma che le usa, si comportano allo stesso modo di quelle fornite dal Basic.

Non è consentito definire una funzione in modo immediato, se si prova si ha il messaggio: ?ILLEGAL DIRECT ERROR

Segue un programma esempio nel quale sono definite due funzioni, e, nella definizione della seconda, viene richiamata la prima.

```
1 REM ES12 PROVA DEF FN
10 DEFFNF1(Y)=Y↑2
20 DEFFNF2(Z)=3*FNF1(Z)
30 INPUT"ARGOMENTO F1: ";A
40 INPUT"ARGOMENTO F2: ";B
50 PRINTFNF1(A),FNF2(B)
60 STOP
```

DIM Tipo: Trattamento interno dati

Formato: DIM var(ind) [,var(ind)...]

dove: var è un nome di variabile nel formato consentito, seguito tra parentesi dai valori massimi di ognuno degli indici, separati da virgole. Gli indici possono essere costanti o variabili numeriche.
Deve essere usata questa istruzione prima di richiamare nel programma le variabili con indice, qualora almeno uno degli indici superi il valore 10. Se non si procede così si ha la segnalazione: ?OUT OF RANGE. L'istruzione DIM deve essere eseguita una sola volta per ogni variabile con indice; se essa viene ripetuta si ha il messaggio: ?REDIM'D ARRAY.
Gli indici partono dal valore 0 e possono teoricamente arrivare a 32767; il numero delle dimensioni di una variabile con indice può arrivare a 255, ma si devono fare i conti con la disponibilità di memoria. Puoi vedere nel Capitolo 8 quanta memoria viene consumata definendo variabili con indice.

Segue un esempio nel quale viene dimensionata una tabella di stringhe A$, formata da 7 righe di 3 colonne ciascuna, 21 elementi in totale. Inoltre viene dimensionata una lista di numeri interi A% che può contenere 100 elementi, e una tabella a 3 dimensioni di numeri con decimali A, nella quale il primo indice varia da 0 a 4, il secondo da 0 a 5 e il terzo da 0 a 6, si hanno 210 elementi.

```
1 REM ES13 PROVA DIM
10 DIMA$(6,2)
20 DIMA%(99),A(4,5,6)
```

Le variabili con indice si richiamano ponendo tra parentesi dopo il nome, delle costanti o delle variabili, che definiscono quale elemento si vuole.

END Tipo: Controllo

Formato: END

Questa istruzione fa terminare l'elaborazione e fa comparire la parola READY. sul video (non compare BREAK ... come con STOP). Il programma può continuare se si scrive CONT e RETURN, sempre che dopo la END siano presenti altre istruzioni. Non ha senso usarla in modo immediato.

EXP Tipo: Funzione matematica

Formato: EXP (numero)

dove numero può anche essere una variabile o un'espressione numerica.

Essa calcola il valore della costante e (base dei logaritmi naturali, numero irrazionale di cui è memorizzata una approssimazione uguale a 2.71828183) elevato alla potenza fornita da numero. Se numero supera 88.0296919 si ha il messaggio: ?OVERFLOW.

Puoi provare il programma che segue per diversi valori di X.

```
1 REM ES15 PROVA EXP
10 INPUT"ESPONENTE PER E: ";X
20 PRINT"E↑";X;"="EXP(X)
30 STOP
```

FN Tipo: Funzione matematica

Formato: FN nome(numero)

dove numero può anche essere una variabile numerica. La funzione con il suo nome è stata definita da una istruzione DEF FN. Se questo non è avvenuto, si ha il messaggio: ?UNDEF'D FUNCTION.
Nella definizione della funzione, cioè nella espressione di calcolo, può anche non comparire la variabile falsa, usata al momento della definizione. In tale caso il valore calcolato non dipende dal numero posto tra parentesi.
Quando viene richiamata una funzione ad essa viene sostituito il suo valore numerico.

Segue un programma nel quale sono definite due funzioni; nella prima la variabile falsa è coinvolta nel calcolo, nella seconda no. Esse sono poi richiamate e viene stampato il risultato.

```
1 REM ES16 PROVA FN
10 DEF FN A(P)=5*P-7
20 DEF FN B(R)=10-A/3
30 PRINT"FN A(6)=";FN A(6)
35 A=2:GOSUB40
37 A=5:GOSUB40
39 STOP
40 PRINT"FN B(0)=";FN B(0)
50 PRINT"FN B(1)=";FN B(1)
55 RETURN
```

RISULTATI PROGRAMMA PRECEDENTE

```
FN A(6)= 23
FN B(0)= 9.33333333
FN B(1)= 9.33333333
FN B(0)= 8.33333333
FN B(1)= 8.33333333


BREAK IN 39
```

Come puoi vedere il valore fornito dalla funzione FN B non dipende dal numero passato tra parentesi, ma dal valore attuale della variabile A coinvolta nel calcolo.

FOR...TO... [STEP...] Tipo: Controllo

Formato: FOR var=lim1 TO lim2 [STEP incr]

dove:

var è il nome di una variabile numerica reale (floating point), che viene usata come contatore (variabile di controllo),

lim1 e lim2 sono il valore iniziale e il valore finale del contatore var,

incr, che esiste solo se è presente la parola chiave STEP, rappresenta l'incremento da usare per il contatore var, per passare dal valore iniziale al valore finale. Se STEP manca l'incremento è assunto implicitamente uguale a 1. STEP può anche essere seguito da incr negativo; in tale caso lim 1 deve essere maggiore di lim 2,

lim1, lim2 e incr possono essere anche espressioni numeriche semplici.

Questa istruzione serve per INIZIARE L'ESECUZIONE di un CICLO e non può essere usata da sola; il ciclo iniziato da FOR e controllato dalla variabile di controllo (contatore) in base al valore iniziale, al valore finale e all'incremento (STEP), termina con la istruzione NEXT var, dove var è lo stesso nome di variabile che compare nella istruzione FOR. Vediamo cosa si intende con "termina"; si intende che vengono eseguite ciclicamente tutte le istruzioni che sono comprese tra la istruzione FOR e la istruzione NEXT collegata alla FOR. Quello che avviene è precisamente questo:

1) con FOR viene inizializzato il contatore, memorizzato il valore finale e l'incremento;
2) vengono eseguite tutte le istruzioni che seguono fino alla istruzione NEXT;
3) viene sommato al contatore l'incremento memorizzato e viene confrontato il valore raggiunto con il valore finale;
4) se il valore raggiunto non supera il valore finale il programma torna alla istruzione immediatamente seguente la istruzione FOR (incomincia un altro ciclo);
5) se il valore raggiunto supera il valore finale il programma prosegue con la istruzione che viene subito dopo la istruzione NEXT, cioè esce dal ciclo.

Le informazioni necessarie per controllare il ciclo FOR sono memorizzate nell'area di lavoro del sistema chiamata STACK; la variabile che serve come contatore sta nella zona variabili del Basic, insieme alle altre variabili.
Se consideri con attenzione questo modo di operare ti accorgi subito che un ciclo FOR viene percorso almeno una volta, infatti l'analisi sul contatore viene fatta quando si incontra il NEXT. Inoltre non sempre viene raggiunto il valore finale; l'incremento può essere tale da far superare lim2, senza raggiungerlo.

Segue un programma esempio, puoi osservare il programma e i suoi risultati e vedrai confermato quanto si è detto. Osserva che all'uscita dal ciclo la variabile di controllo supera lim2 per incremento positivo e gli è inferiore per incremento negativo.

```
1 REM ES17 PROVA FOR...TO...STEP
5 PRINT"ESECUZIONE CON FOR"
10 FORK=1TO10
20 PRINTK;" ";:NEXTK:PRINT
23 PRINT"VALORE K ALL'USCITA =";K
25 PRINT:PRINT"ESECUZIONE SENZA FOR"
30 K=1
35 PRINTK;" ";:K=K+1:IFK<=10THEN35
40 PRINT:PRINT
50 PRINT"CICLO CON INCREMENTO NEGATIVO"
60 FORK=10TO1STEP-1:PRINTK;" ";:NEXTK
```

```
65 PRINT:PRINT"VALORE K ALL'USCITA =";K
70 PRINT:PRINT"CICLO CON INCREMENTO <1"
75 FORK=1TO3STEP.3:PRINTK;" ";:NEXTK
77 PRINT:PRINT"VALORE K ALL'USCITA =";K
```

RISULTATI PROGRAMMA PRECEDENTE

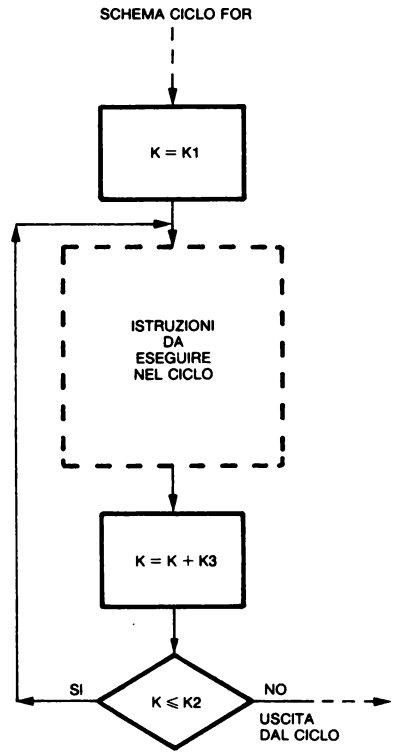
```
ESECUZIONE CON FOR
 1   2   3   4   5   6   7   8   9   10
VALORE K ALL'USCITA = 11

ESECUZIONE SENZA FOR
 1   2   3   4   5   6   7   8   9   10

CICLO CON INCREMENTO NEGATIVO
 10   9   8   7   6   5   4   3   2   1
VALORE K ALL'USCITA = 0

CICLO CON INCREMENTO <1
 1   1.3   1.6   1.9   2.2   2.5   2.8
VALORE K ALL'USCITA = 3.1
```

Segue lo schema a blocchi di un ciclo FOR.



FOR K=K1 TO K2 STEP K3

I cicli FOR devono essere usati bene; non si deve uscire da un FOR senza averlo chiuso ponendo la variabile di controllo del ciclo al valore finale e facendo "scattare" il NEXT.
Considera l'esempio che segue:

```
1 REM ES17BIS PROVA FOR
10 DATA13,90,67,456,-987,1234,675
11 DATA1,95,-567,-980,567,23456,689,32
15 FORK=1TO15:READX:PRINTX;:NEXT:PRINT
20 INPUT"NUMERO: ";N
25 RESTORE:FORK=1TO15
30 READX
35 IFX=NTHENPRINT"TROVATO UGUALE":K=15:NEXT:STOP
40 NEXT:PRINT"NON TROVATO UGUALE":STOP
```

in esso si esce dal FOR per una condizione intermedia, ma si opera correttamente. Infatti sia che si esca per esaurimento del ciclo, che per valore trovato uguale, si fa scattare il NEXT. La presenza di due NEXT per un solo FOR non dà fastidio, infatti sono usati in alternativa.
I cicli FOR possono essere concatenati, cioè posti uno interno all'altro, fino ad un massimo di 9.

FRE Tipo: Funzione lettura memoria

Formato: FRE (var)

dove: var può essere un numero o una variabile di qualunque tipo, che non influisce sul risultato della funzione, però deve essere presente.

La funzione fornisce il numero di byte di memoria RAM liberi. Se tale numero è negativo devi sommargli 65536 per ottenere un risultato corretto.

Segue un programma che fornisce il numero dei byte liberi all'inizio, assegna 101 variabili stringa, e poi fornisce di nuovo il numero dei byte liberi. La differenza tra i due risultati dà il numero dei byte usati per le variabili del programma. Nota il modo di calcolare la memoria libera alla linea 100 (sottoprogramma): FRE(0)<0 è una relazione logica che ha valore -1 quando è vera, per cui se FRE(0) risulta negativo si ha che (FRE(0)<0)=—1; in conseguenza 65536 viene sommato al primo FRE(0) e il risultato risulta positivo.

```
1 REM ES18 PROVA FRE
10 GOSUB100:PRINT"BYTE LIBERI INIZIO:";Z
20 DIMA$(100):Z$="PROVA FRE"
30 FORK=0TO100:A$(K)=Z$:NEXTK
40 GOSUB100
50 PRINT"BYTE LIBERI DOPO ASS. VAR.:";Z
60 STOP
100 Z=FRE(0)-(FRE(0)<0)*65536:RETURN
```

RISULTATI PROGRAMMA PRECEDENTE

```
BYTE LIBERI INIZIO: 38714
BYTE LIBERI DOPO ASS. VAR.: 38390

BREAK IN 60
```

Se provi ad eseguire in immediato FRE(0)+ 65536 subito dopo aver acceso il calcolatore trovi il valore 38909.

GET Tipo: Operazione I/O dati

Formato: GET lista-var

dove lista-var è una lista di nomi di variabili separate da virgola, al limite una sola variabile.

Questa istruzione legge un carattere per volta dal buffer della tastiera. Il buffer può contenere 10 caratteri, se si pigiano più di 10 tasti, gli ultimi caratteri vanno persi. Quando si legge con GET un carattere, si crea un posto nel buffer, cioè il carattere letto viene eliminato.
Segue un programma esempio:

```
1 REM ES19 PROVA GET
10 PRINT"SCRIVI UNA PAROLA E"
11 PRINT"TERMINA CON ASTERISCO"
15 FORK=1TO3000:NEXTK
20 GETA$:PRINTA$;
30 IFA$="*"THEN50
40 GOTO20
50 PRINT"FINITO PRIMA PROVA"
60 PRINT"ASPETTO UN TASTO PER PROSEGUIRE"
70 GETA$:IFA$=""THEN70
80 PRINT"PROSEGUO"
90 PRINT"PREMI S PER CONTINUARE, N PER STOP"
93 GETA$:IFA$=""THEN93
95 IFA$<>"S"ANDA$<>"N"THEN93
97 IFA$="S"THEN10
99 STOP
```

La prima prova consiste in questo (linee da 10 a 50):
.. in 10 e 11 viene chiesto di scrivere una parola e di terminare con asterisco; tale parola deve essere corta, cioè entrare nel buffer, in tutto meno di 10 caratteri; vedrai che mentre premi i tasti i caratteri non appaiono;
.. in 15 viene creato un ciclo di attesa, è proprio durante il ciclo nel quale il calcolatore percorre un FOR 3000 volte che tu scrivi e riempi il buffer;
.. in 20, terminato il ciclo, viene prelevato un carattere con GET dal buffer e viene stampato;
.. in 30 e 40 se il carattere stampato è asterisco il programma prosegue da 50, altrimenti torna a 20;

.. da 60 a 80 il programma prosegue solo se si preme un tasto, la istruzione GET A$ della linea 70 lo blocca; ecco un modo per creare una pausa lunga a piacere;
.. a 90 viene chiesto di premere uno tra due tasti, S o N;
.. a 93 GET A$ legge un tasto e prosegue solo quando il tasto viene premuto, infatti se non è premuto alcun tasto A$ è la stringa nulla;
.. a 95 viene controllato il tasto premuto e se non è uno dei due richiesti il programma torna a 93;
.. a 97 se A$ è uguale a S il programma ricomincia, altrimenti si ferma.

Già da questo breve programma hai visto diversi utilizzi di GET. Nel Capitolo 3 vedrai altre applicazioni.
Tieni presente che è meglio far seguire a GET una variabile di tipo stringa, infatti se usi il nome di una variabile numerica e poi premi un tasto non numerico, il sistema segnala un errore.

Questa istruzione non può essere usata in modo immediato.

GET# Tipo: Operazione I/O dati

Formato: GET# lfn, lista-var

dove:
lfn è il numero logico di un file che deve essere stato preventivamente aperto con OPEN;
lista-var è una lista di nomi di variabili separate da virgola, al limite una sola variabile.

Con questa istruzione viene letto un carattere per volta da una periferica diversa dalla tastiera; se non viene ricevuto un carattere e la variabile è di tipo stringa essa diventa una stringa nulla, mentre se è di tipo numerico essa riceve un valore 0.

Segue un programma esempio, nel quale all'inizio (linee da 10 a 30) viene pulito il video e vengono scritte le seguenti 3 righe:

APERTURA VIDEO COME FILE # 1
STO PROVANDO A LEGGERE DAL VIDEO
*

```
1 REM ES20 PROVA GET#
10 PRINT"□ APERTURA VIDEO COME FILE #1"
20 PRINT"STO PROVANDO A LEGGERE DAL VIDEO"
25 PRINT"*"
27 OPEN1,3:REM APRE DEVICE 3 COME FILE 1
```

```
30 PRINT"S";:Z$=""
40 GET#1,A$:Z$=Z$+A$
45 IFA$="*"THEN60
50 GOTO40
60 PRINT"SDDDDDDDDD";Z$
65 CLOSE1:STOP
```

poi, alla linea 27 viene aperto il video che è la periferica numero 3, come file logico numero 1. Alla linea 30 con PRINT del carattere HOME si porta il cursore nell'angolo alto a sinistra senza cancellare il video, e si annulla la stringa Z$. Dalla linea 40 alla 50 vengono letti i caratteri delle prime righe del video, uno dopo l'altro con GET# 1, e sommati in Z$. La lettura termina dopo aver incontrato l'asterisco. Alla linea 60 viene portato il cursore in giù e viene stampata la stringa Z$ che contiene tutti i caratteri letti con GET# 1.

Il comando GET# è importante perchè legge tutti i caratteri, anche quelli di controllo, come le virgole separatrici dei campi o il carattere corrispondente al RETURN, CHR$(13).

Questo comando non può essere usato in modo immediato.

GOSUB Tipo: Controllo

Formato: GOSUB numero-linea

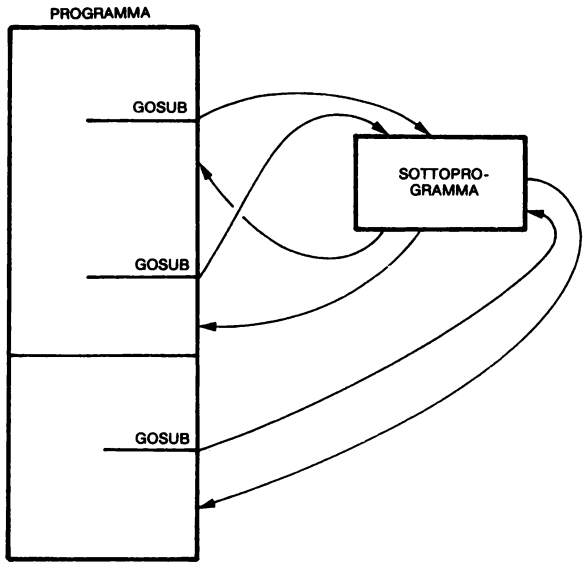
dove numero-linea è il numero di una linea, dove inizia un sottoprogramma.

Questa istruzione trasferisce il controllo del programma alla linea indicata (che rappresenta l'inizio di un sottoprogramma) e ricorda, cioè memorizza, il numero della linea in cui è il GOSUB. Il numero di linea che è stato memorizzato viene riesumato, quando nel sottoprogramma si incontra l'istruzione RETURN (che non ha relazione con il tasto RETURN della tastiera) e serve per ritornare all'istruzione subito dopo la GOSUB, riprendendo la sequenza interrotta.

Per sottoprogramma si intende un pezzo di programma che esegue una ben definita sequenza di operazioni e termina con una istruzione RETURN. La programmazione che fa uso di sottoprogrammi risulta molto ordinata e di facile comprensione; infatti ogni sottoprogramma, in genere poche istruzioni, ha una funzione e fornisce risultati ben definiti. Richiamando un sottoprogramma già collaudato, dove serve, si è sicuri di non fare errori, e, inoltre si risparmia molta memoria.
La differenza che esiste tra le funzioni e i sottoprogrammi è che una funzione consiste in una sola formula, mentre un sottoprogramma è formato in generale da più istruzioni.

Segue uno schema a blocchi che esemplifica la chiamata di un sottoprogramma e il ritorno da un sottoprogramma.

SCHEMA GOSUB



Segue un semplice esempio di uso di sottoprogrammi:

```
1 REM ES21 PROVA GOSUB
10 PRINT"PROSPETTO ALUNNI CLASSE III A"
20 GOSUB100
25 PRINT"ISCRITTI NUMERO 25"
30 GOSUB110:GOSUB100
35 PRINT"PROMOSSI NUMERO 25"
40 C$="\":GOSUB120
45 C$="/":GOSUB120
99 STOP
100 FORK=1TO40:PRINT"-";:NEXTK:RETURN
110 FORK=1TO40:PRINT"*";:NEXTK:RETURN
120 FORK=1TO40:PRINTC$;:NEXTK:RETURN
```

Alla linea 100 e alla linea 110 sono presenti due sottoprogrammi che tracciano rispettivamente una linea di trattini e una linea di asterischi. Essi sono richiamati in

20 e 30. Alla linea 120 è presente un sottoprogramma che riceve dal programma in C$ il carattere da usare per tracciare la linea.

La memorizzazione del numero di linea, e anche della posizione dell'istruzione nella linea per le istruzioni multiple, ha luogo in una speciale area di lavoro chiamata STACK, di 256 byte. Il meccanismo di utilizzo dell'area STACK è il seguente:
.. al momento del GOSUB viene memorizzata l'informazione per il ritorno come ultima nella pila di informazioni già memorizzate;
.. al momento del RETURN viene prelevata l'informazione che si trova sopra tutte le altre, che risulta essere l'ultima depositata (in inglese si dice LAST IN FIRST OUT, l'ultima messa è la prima tolta);
.. questa gestione consente che un sottoprogramma ne chiami un altro, fino ad un massimo di 23 (concatenamento dei sottoprogrammi).

Segue il programma GOSUBCONC, che dimostra quale è il numero massimo di GOSUB concatenati consentiti. Infatti fino a quando non si esegue il RETURN l'area STACK non viene svuotata, a un certo punto manca spazio e si ha il messaggio di errore: OUT OF MEMORY.

```
1 REM GOSUBCONC
10 K=0:GOSUB100:STOP
100 :K=K+1:PRINTK;:GOSUB100
```

GOTO Tipo: Controllo

Formato: GOTO numero-linea oppure
GO TO numero-linea

Questa istruzione fa proseguire il programma dal numero di linea indicato.
Se il numero di linea non è presente nel programma si ha la segnalazione: ?UNDEF'D STATEMENT ERROR IN .. .
Scrivendo:

```
120 GOTO 120
```

si crea un ciclo senza fine che può essere interrotto premendo il tasto RUN/STOP.

IF...THEN... Tipo: Controllo

Formato: IF espressione THEN numero-linea
IF espressione GOTO numero-linea
IF espressione THEN istruzione

dove:

espressione è una qualunque espressione valida nel linguaggio che può essere VERA o FALSA;

la parte di istruzione che viene dopo THEN o GOTO viene eseguita se il risultato di espressione è VERO;

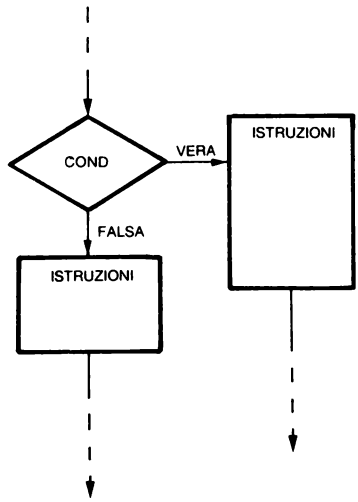
se il risultato di espressione è FALSO, il programma prosegue dal numero-linea immediatamente successivo a quello della linea che contiene l'istruzione IF.

Se si usa il formato con GOTO, la condizione VERA provoca un salto in un determinato punto del programma; siamo in presenza di un salto sotto condizione. La normale istruzione GOTO provoca invece un salto incondizionato.
Il formato THEN numero-linea è sostanzialmente identico al GOTO numero-linea.
Quello che presenta maggiore interesse, in certi casi naturalmente, è il formato THEN istruzione. Infatti dopo il THEN possono essere presenti più istruzioni separate dal carattere separatore due punti. Devi fare attenzione al fatto che se la serie di istruzioni non termina con un GOTO, dopo viene eseguita la stessa sequenza del caso di espressione FALSA.
Naturalmente in questo caso nelle istruzioni dopo THEN può comparire un'altra istruzione IF e bisogna considerare con attenzione il concatenamento delle condizioni VERO e FALSO.

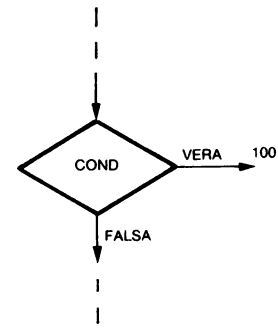
Segue un programma esempio nel quale sono riportati i tre formati dell'istruzione IF.

```
1 REM ES23 PROVA IF
10 INPUT"SCRIVI UN NUMERO: ";N
20 IFN<0THEN40
30 PRINT"NUMERO POSITIVO":GOTO50
40 PRINT"NUMERO NEGATIVO"
50 IFABS(N)>=500GOTO70
60 PRINT"NUMERO MINORE 500":GOTO80
70 PRINT"NUMERO MAGGIORE/UGUALE 500"
80 INPUT"SCRIVI UNA PAROLA: ";A$
90 PRINTA$
95 IFLEN(A$)>10THENPRINT"LUNGH. A$ >10":STOP
97 PRINT"LUNGH. A$ <= 10"
100 STOP
```

Seguono gli schemi a blocchi dei diversi tipi di IF.



IF COND THEN ISTRUZ.



IF COND THEN 100
IF COND GOTO 100

INPUT Tipo: Operazione I/O dati

Formato: INPUT ["messaggio";] lista-var

dove:

messaggio può essere presente e in questo caso deve essere una stringa delimitata dalle virgolette e seguita dal punto e virgola (;); non è ammessa una variabile stringa;

lista-var è una lista di variabili separate da virgola, al limite una variabile sola.

Essa serve per leggere dati dalla tastiera e assegnarli alle variabili.
Rispondendo solo con il tasto RETURN alla richiesta di dato, la variabile coinvolta mantiene il suo precedente contenuto. Per scrivere un dato devi digitarlo e poi premere il tasto RETURN.
L'istruzione provoca la comparsa di un punto interrogativo sul video, per chiedere dati. Devi rispondere digitando i dati e separandoli con virgole se sono più di uno; il RETURN finale chiude il colloquio. Se rispondi con meno dati di quelli richiesti, compaiono due punti interrogativi a significare che non hai soddisfatto la richiesta in modo completo. Se usi come separatore i due punti, viene accettato un dato fino ai due punti e il resto viene ignorato, sia per dati numerici che per stringhe. Se invece usi il punto e virgola viene segnalato errore:?REDO FROM START per i dati numerici e viene richiesto il dato di nuovo; per le stringhe invece il punto e virgola è

accettato come carattere della stringa. Se vuoi comprendere nelle stringhe anche caratteri separatori altrimenti non accettati devi aprire le virgolette prima di iniziare la stringa e chiuderle subito dopo.
Se rispondi con dati stringa a richieste numeriche (accetta solo: segno, cifre e un punto decimale) hai il messaggio: ?REDO FROM START e ti viene chiesto di nuovo il dato.
La lunghezza del messaggio, se presente, non può superare 39 caratteri, infatti il punto interrogativo che compare a chiedere dati non deve passare a nuova linea, altrimenti esso viene incorporato nella risposta. Comunque tra messaggio e risposta non possono essere superate due linee del video, compreso il RETURN finale.
Nel caso tu voglia avere un messaggio esplicativo più lungo, puoi scriverlo con PRINT e poi usare INPUT solo per il dato. Se rispondi con più dati di quelli richiesti, compare il messaggio:?EXTRA IGNORED e il programma prosegue; i dati in più non sono stati trasferiti in alcuna variabile e vanno persi.
L'istruzione INPUT non può essere interrotta, cioè non sente il tasto RUN/STOP. Se desideri interrompere una richiesta di dati e fermare il programma, devi premere RUN/STOP e contemporaneamente RESTORE.
Questa istruzione non può essere usata in modo immediato.

Segue un programma con il quale puoi verificare quanto si è detto.

```
1 REM ES24 PROVA INPUT
10 INPUTA:PRINTA
20 INPUTA%:PRINTA%
30 INPUTA$:PRINTA$
40 INPUTA,B,C
50 PRINTA,B,C
60 INPUT"SCRIVI UN NUMERO: ";N
70 PRINTN
80 INPUT"SCRIVI UNA FRASE: ";A$
90 PRINTA$
99 STOP
```

Nel Capitolo 3 torniamo diffusamente sull'argomento dell'ingresso e dell'uscita dei dati, colloquio video/tastiera.

INPUT# Tipo: Operazione I/O dati

Formato: INPUT# lfn, lista-var

dove:
lfn è il numero logico di un file che deve essere stato aperto su una periferica con il comando OPEN;
lista-var è una lista di variabili separate da virgole, al limite una variabile sola.

L'istruzione preleva dati dal file considerando terminata una variabile quando incontra un separatore che può essere:
.. il carattere generato dal tasto RETURN, cioè CHR$(13);
.. la virgola;
comunque una variabile non può essere più lunga di 80 caratteri, compreso il carattere separatore.
Ovviamente un numero non ha mai più di 80 cifre, mentre una stringa può avere più di 80 caratteri. Se si legge una stringa troppo lunga si ha il messaggio: ?STRING TO LONG, mentre se si legge in una variabile numerica una stringa alfanumerica, si ha il messaggio:?BAD DATA.
Una stringa troppo lunga, o anche un numero troppo lungo si può trovare in un file, se questo è stato scritto in modo errato; vai a vedere l'istruzione PRINT# .

Questa istruzione non può essere usata in modo immediato.

Segue un programma esempio, che risulta dalla modifica di quello riportato nella spiegazione dell'istruzione GET# . Qui si apre il video come file e si va a leggere con l'istruzione INPUT# .

```
1 REM ES25 PROVA INPUT#
10 PRINT"▒APERTURA VIDEO COME FILE #1"
20 PRINT"STO PROVANDO A LEGGERE DAL VIDEO"
25 PRINT"*"
27 OPEN1,3:REM APRE DEVICE 3 COME FILE 1
30 PRINT"▒";:Z$=""
40 INPUT#1,A$
45 INPUT#1,B$
50 INPUT#1,C$
60 PRINT"▒▒▒▒▒▒▒▒▒▒"
63 PRINTA$,B$,C$
65 CLOSE1:STOP
```

I comandi relativi ai file su cassetta e su floppy disk sono trattati diffusamente nel secondo volume di questa serie di manuali per il COMMODORE 64.

INT Tipo: Funzione matematica

Formato: INT (numero)

dove numero può essere un numero, una variabile numerica o una espressione numerica.

La funzione fornisce la parte intera dei numeri positivi, troncando i decimali, mentre per i numeri negativi con decimali viene fornito il numero negativo immediatamente inferiore (INT(-25.456)=-26).

Segue un programma esempio dove si calcola il resto di una divisione tra due numeri.

```
1 REM ES26 PROVA INT
10 INPUT"PRIMO NUMERO:    ";N1
20 INPUT"SECONDO NUMERO: ";N2
30 N3=N1/N2
40 R=N3-INT(N3)
50 PRINT"PARTE INTERA N1/N2: ";INT(N3)
55 PRINT"PARTE DECIMALE N1/N2: ";N3-INT(N3)
60 PRINT"RESTO DIV. SENZA DEC.: ";N1-N2*INT(N3)
70 PRINT"N1/N2: ";N3
80 STOP
```

LEFT$ Tipo: Funzione stringa

Formato: LEFT$ (stringa, intero)

dove:
stringa è una stringa o una variabile stringa (anche la somma di più stringhe e/o variabili stringa);
intero è un numero intero o una variabile numerica con valore compreso tra 0 e 255.

La funzione fornisce i primi "intero" caratteri della stringa se essa è più lunga, tutta la stringa se essa è più corta.

Provando il programma esempio che segue, puoi introdurre diverse risposte per A$ e studiare il comportamento della funzione.

```
1 REM ES27 PROVA LEFT$
5 S$="                    "
10 INPUT"SCRIVI UNA FRASE: ";A$
20 B$=LEFT$(A$+S$,20)
25 C$=LEFT$(A$,3)
30 PRINTB$;"*"
35 PRINTC$
40 STOP
```

Alla linea 20, sommando ad A$ la stringa S$ di 20 spazi, si è sicuri che la B$ risulta di 20 caratteri. Infatti se si opera con la funzione su una stringa più corta della lunghezza richiesta, la stringa risultato non ha la lunghezza richiesta. Alla linea 30 si stampa un asterisco subito dopo B$ per metterne in evidenza la lunghezza anche se contiene spazi.

LEN Tipo: Funzione lettura memoria

Formato: LEN (stringa)

dove stringa può essere una stringa o una variabile stringa.

Essa fornisce il numero dei caratteri che compongono la stringa.

Segue un semplicissimo programma per provare la funzione.

```
1 REM ES28 PROVA LEN
10 INPUT"STRINGA: ";A$
20 PRINTLEN(A$)
30 STOP
```

LET Tipo: Trattamento interno dati

Formato: [LET] var=espressione

dove: LET può essere omessa;
var è il nome di una variabile consentita dal Basic;
espresssione è una espressione compatibile con il tipo della variabile che compare a sinistra dell'uguale.

Questa istruzione serve per assegnare dati alle variabili. Se si tratta di variabili numeriche, espressione può essere un semplice numero o una complicata espressione di calcolo. Se si tratta di variabili stringa, espressione può essere una stringa delimitata da virgolette, una variabile stringa o una somma di stringhe e/o variabili stringa.

Segue un programma nel quale si fa vedere che usando e non usando la parola chiave LET si ottengono gli stessi risultati.

```
1 REM ES29 PROVA LET
10 INPUT"PRIMO NUMERO    :";N1
20 INPUT"SECONDO NUMERO :";N2
```

```
30 LETM=N1+N2-(N1/(2+INT(N2)))
40 P=N1+N2-(N1/(2+INT(N2)))
45 PRINT"M=";M:PRINT"P=";P
50 STOP
```

LIST Tipo: Servizio

Formato: LIST [[prima.linea] [—ultima.linea]]

Serve per listare sul video il programma presente in memoria.

LIST senza numeri di linea, lista tutto il programma; si ha lo scrolling del video durante la lista, per rallentare il movimento si può tenere premuto il tasto CTRL

LIST	numero.linea	lista una sola linea
LIST	numero.linea-	lista a partire da numero.linea
LIST	-numero.linea	lista fino a numero.linea, partendo dall'inizio
LIST	numero1-numero2	lista dalla linea numero1 alla linea numero2.

Se vuoi fermare la lista devi premere il tasto RUN/STOP.

Usando le istruzioni OPEN e CMD si può dirigere la lista verso un'altra periferica, in particolare verso la stampante.

Questa istruzione può essere usata anche all'interno di un programma, solo che, alla fine della lista richiesta, non restituisce più il controllo al programma. Se premi CONT, ottieni ancora la lista del programma.

LOAD Tipo: Operazione I/O programmi

Formato: LOAD [fn] [,dn] [,ind]

dove:

fn (stringa tra virgolette) è il nome del file da caricare in memoria; può anche essere il nome di una variabile stringa che contiene il nome del file;

dn è il numero logico della periferica da cui prelevare il programma;

ind è un numero che fornisce informazioni sull'indirizzo di memoria da cui iniziare il caricamento del file.

Serve per caricare programmi in memoria prelevandoli da nastro o da disco. Il nastro è di norma individuato da dn=1, il disco da dn=8.
Di solito i programmi Basic vengono caricati in memoria a partire dall'indirizzo 2049, ponendo 1 per ind, si può caricarli partendo dall'indirizzo dal quale partivano prima di essere memorizzati con il comando SAVE. Il puntatore all'inizio del programma si trova nella pagina 0 della RAM nei byte 43 e 44 e può essere modificato.
Con i comandi LOAD, SAVE e VERIFY, che riguardano operazioni con i programmi, non si deve usare prima l'istruzione OPEN.
Se si usa un nome di programma inesistente, si ha il messaggio: ?FILE NOT FOUND.
Elenchiamo i possibili formati di LOAD.

LOAD senza parametri, legge il primo programma che trova sul nastro.

LOAD N$ oppure LOAD "nome"
legge da nastro il programma con il nome fornito.

LOAD "",1,1 oppure LOAD n$,1,1 oppure LOAD "nome",1,1
legge da nastro il primo programma che trova o il programma avente il nome fornito, ma lo carica a partire dall'indirizzo dal quale iniziava quando è stato memorizzato (non opera una rilocazione del programma).

LOAD "*",8 legge da disco il primo programma, cioè quello che compare per primo nella DIRECTORY del disco.

LOAD "nome",8 oppure LOAD n$,8
legge da disco il programma con il nome fornito.

LOAD "nome",8,1 oppure LOAD n$,8,1 legge da disco il programma con il nome fornito senza rilocarlo.

Quando si dà il comando LOAD si ha un colloquio tramite video, e precisamente:

.. per il nastro:
PRESS PLAY ON TAPE
FOUND nome...
LOADING
READY.

se il programma non è il primo, si ha una ripetizione di FOUND... fino ad arrivare al nome cercato; dopo il primo messaggio si deve avviare il registratore premendo il tasto PLAY;

.. per il disco:
SEARCHING FOR nome
LOADING
READY.

L'istruzione LOAD può essere data in modo immediato; in questo caso il sistema prima di caricare il programma chiude tutti i file aperti e azzera tutte le variabili. Quando si usa LOAD dall'interno di un programma, il nuovo programma caricato, parte automaticamente; in questo modo si possono concatenare tra loro più programmi e non vengono azzerate le variabili, ma si deve fare attenzione alla lunghezza dei programmi per non perdere variabili a causa di sovrapposizioni. Vedi il Capitolo 1 del secondo Volume in merito.

LOG Tipo: Funzione matematica

Formato: LOG (argomento)

dove argomento può essere un numero o una espressione numerica.

Calcola il logaritmo naturale, cioè in base e, dell'argomento.
L'argomento deve essere diverso da zero e positivo, altrimenti si ha il messaggio: ?ILLEGAL QUANTITY.

Segue un esempio; puoi provare a calcolare diversi logaritmi.

```
1 REM ES32 PROVA LOG
5 A$="LOG(":B$=") IN BASE 10="
10 INPUT"NUMERO: ";N
20 PRINT"LOG(";N;")=";LOG(N)
30 PRINTA$;N;B$;LOG(N)/LOG(10)
40 PRINT"LOG(2.71828183)="LOG(2.71828183)
50 PRINT"LOG(57/3+78)=";LOG(57/3+78)
60 STOP
```

MID$ Tipo: Funzione stringa

Formato: MID$ (stringa, int-1 [,int-2])

dove:
stringa può essere una stringa delimitata da virgolette, una variabile stringa o una somma di stringhe e/o di variabili stringa;
int-1 è un intero o una variabile intera che indica da quale carattere della stringa

(dà la posizione) iniziare l'estrazione della sottostringa;
int-2, se presente indica quanti caratteri estrarre; se int-2 manca si intende che si arriva alla fine della stringa.

Serve per estrarre parti di stringa da una stringa. Se int-1 manda fuori dalla stringa o int-2 è uguale a zero, la stringa estratta è la stringa nulla. Se int-2 è troppo grande vengono estratti meno caratteri.

Segue un programma esempio:

```
1 REM ES33 PROVA MID$
5 R$="4 CARATTERI CENTRALI: "
10 INPUT"STRINGA: ";A$
20 L=LEN(A$)
25 IFL<4THENSTOP
30 PRINT"PRIMI 3 CARATTERI: ";MID$(A$,1,3)
40 PRINT"ULTIMI 3 CARATTERI: ";MID$(A$,L-2)
50 PRINTR$;MID$(A$,L/2-1,4)
60 STOP
```

Ti viene chiesta una stringa, viene calcolata la lunghezza della stringa e vengono stampati i primi tre caratteri, gli ultimi tre caratteri e i quattro caratteri centrali.

NEW Tipo: Servizio

Formato: NEW

Serve per cancellare il programma Basic presente in memoria e tutte le sue variabili. Si deve usare prima di iniziare la scrittura di un programma. Si può usare anche come istruzione di programma, ma quando viene eseguita il programma che la contiene viene cancellato e quindi non va oltre questa istruzione.

NEXT Tipo: Controllo

Formato: NEXT var [,var]...

dove var sono nomi di variabili inizializzate come contatori in un ciclo FOR.

Serve per delimitare il campo di azione di un ciclo FOR. Quando viene incontrato NEXT, il contatore di ciclo viene incrementato (o decrementato) e confrontato con il limite per stabilire se continuare il ciclo o uscirne.
I cicli FOR possono essere concatenati tra loro fino a 9 volte, cioè 9 cicli uno interno all'altro. La parola chiave NEXT può anche comparire senza il nome della variabi-

le, in questo caso viene riferita all'ultimo ciclo FOR aperto. Si ha invece un messaggio di errore: ?NEXT WITHOUT FOR, se si cita in modo errato la variabile. Inoltre se più cicli FOR concatenati si chiudono insieme, si può scrivere una sola parola chiave NEXT seguita dai nomi delle diverse variabili coinvolte separate da virgola; naturalmente i nomi delle variabili devono essere scritti nel giusto ordine, prima quella del FOR aperto per ultimo e così via.

Segue un programma esempio dove puoi vedere applicate alcune delle regole esposte.

```
1 REM ES35 PROVA NEXT
10 PRINT"TABELLA DI NUMERI■"
20 FORK=1TO10
23 FORJ=1TO6
24 X=K*J:X$=STR$(X):IFLEN(X$)<3THENX$=" "+X$
26 PRINTX$;"  ";
27 NEXTJ:PRINT
29 NEXTK
30 PRINT"■RIEMPIE MATRICE A 3 DIMENSIONI■"
31 DIMZ(3,4,5)
35 FORK=0TO3:FORJ=0TO4:FORL=0TO5
40 Z(K,J,L)=K*J*L
45 NEXTL,J,K
50 INPUT"QUALE ELEMENTO (K,J,L):";K,J,L
55 PRINT"Z(";K;",";J;",";L;")=";Z(K,J,L)
60 STOP
```

Nota alla linea 24 l'accorgimento usato per incolonnare i numeri con diverso numero di cifre. Se vuoi vedere altri elementi della matrice Z(k,j,l), puoi scrivere in immediato GOTO 50.

ON Tipo: Controllo

Formato: ON espressione GOTO/GOSUB lista-numeri-linea

dove:

espressione può essere una qualunque espressione numerica che dia un risultato non negativo da 1 in avanti del quale viene considerata solo la parte intera;

lista-numeri-linea è una lista di numeri di linea presenti nel programma, separati da virgole.

Serve per saltare a linee di programma con GOTO, a sottoprogrammi con GOSUB,

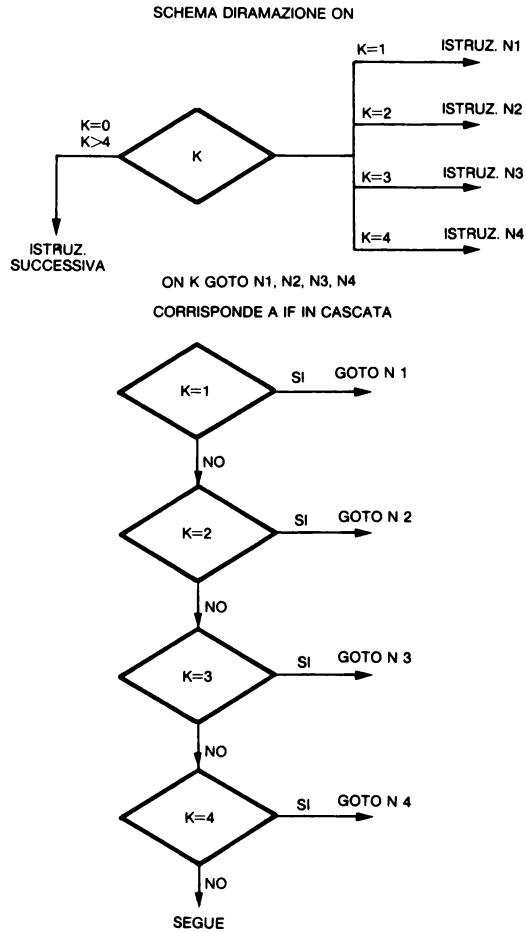
a seconda del valore numerico di espressione. Se il valore è 1, il salto avviene al primo numero di linea della lista, se esso è 2, al secondo, se esso è N, all'ennesimo. Se il valore è zero o supera il numero di numeri di linea presenti, il programma prosegue dalla linea di programma seguente. Un valore negativo dell'espressione provoca il messaggio: ?ILLEGAL QUANTITY.

Questa istruzione sostituisce, se si trova l'algoritmo adatto per scrivere l'espressione, una catena di IF...THEN.

Segue un programma esempio.

```
1 REM ES37 PROVA ON
10 INPUT"SCEGLI: A,B,C,D ";X$
15 IFASC(X$)>=65ANDASC(X$)<=68THEN20
17 GOTO10
20 ONASC(X$)-64GOTO100,200,300,400
100 PRINT"SCELTO A":STOP
200 PRINT"SCELTO B":STOP
300 PRINT"SCELTO C":STOP
400 PRINT"SCELTO D":STOP
```

Segue un diagramma esplicativo dell'istruzione ON.



OPEN Tipo: Operazione I/O dati

Formato: OPEN lfn,dn [,sa] [,"fn [,ft] [,md] "]

dove i parametri hanno il significato spiegato all'inizio di questo paragrafo.

Questa istruzione serve per aprire un file logico, assegnandogli il numero lfn, sulla periferica contraddistinta dal numero dn. Questi primi due parametri devono essere presenti dopo la parola chiave OPEN, se il secondo è omesso, esso viene

assunto uguale a 1. Gli altri possono anche mancare, e, a seconda della periferica, possono avere diversi significati.
Il numero lfn è citato in tutte le operazioni di I/O relative al file. Esso può variare da 1 a 255, ma solitamente si usano numeri al disotto di 127, in quanto gli altri possono avere speciali significati per particolari periferiche.
Il numero dn per le periferiche usuali è riportato nella tabellina posta all'inizio di questo paragrafo.
Il numero sa, se presente, specifica ulteriormente il tipo di operazione per la quale si apre il file.
Il nome del file non è sempre necessario.

Quando viene omesso il dn, esso viene assunto uguale a 1 e quindi la OPEN è verso il registratore a nastro. Per questa periferica i valori di sa sono:
.. 0 per operazioni di lettura (INPUT...)
.. 1 per operazioni di scrittura senza segnalazione di fine nastro
.. 2 per operazioni di scrittura con segnalazione di fine nastro (end-of-tape); questa segnalazione di fine nastro che viene registrata, fa sì che non si possa, per errore, andare a leggere dopo la fine del file, cosa che provoca il messaggio: ?DEVICE NOT PRESENT.
Per operazioni su nastro è bene che nella OPEN sia presente anche il nome del file fn.
Nel caso delle operazioni su disco, dn è uguale a 8 (se è presente una sola unità); sa può variare tra 2 e 14 ed è necessario il nome del file fn. Per informazioni complete sulle operazioni disco, si rimanda ai successivi volumi.

Per le operazioni sulla stampante si rimanda al Capitolo 2 del secondo Volume.

Il nome del file, fn, è una stringa di al massimo 16 caratteri; può essere una variabile.

Se non si apre un file e si usano le altre operazioni di I/O si ha il messaggio: ?FILE NOT OPEN. Se si apre un file già aperto si ha il messaggio: ?FILE OPEN. Se si apre un file per leggere ed esso non esiste si ha il messaggio: ?FILE NOT FOUND. Se si apre un file per scrivere ed esso esiste già (periferica disco solo) si ha il messaggio: ?FILE EXISTS.

Riportiamo qui solo pochi esempi di OPEN, rimandando per gli altri ai capitoli o volumi relativi alle periferiche.

OPEN1,3 apre il video come Input o come Output

OPEN2,0 apre la tastiera come Input

Segue il programma VIDEOFILE, nel quale si apre il video come file e si eseguono operazioni di lettura e scrittura.

```
1 REM VIDEOFILE
10 OPEN3,3
15 PRINT#3,"[illegible]";"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
17 A$=""
20 PRINT#3,"[illegible]";
23 FORK=1TO26:GET#3,B$
25 A$=A$+B$
30 NEXTK:PRINT#3
35 PRINT#3,"[illegible]"A$
40 CLOSE3:STOP
```

PEEK Tipo: Funzione lettura memoria

Formato: PEEK (indirizzo)

dove indirizzo deve essere un numero tra 0 e 65535, cioè un indirizzo di memoria compreso in 64K.

Serve per leggere contenuti dei byte della memoria. Il numero letto varia da 0 a 255. Indirizzi fuori dai limiti provocano il messaggio: ?ILLEGAL QUANTITY.

Segue un programma che legge alcuni puntatori e alcuni byte.

```
1 REM ES40 PROVA PEEK
5 A=PEEK(43)+256*PEEK(44)
10 PRINT"PUNTATORE INIZIO BASIC :";A
15 B=PEEK(45)+256*PEEK(46)
20 PRINT"PUNTATORE INIZIO VARIABILI :";B
30 PRINT"CONTENUTO BYTE 0: ";PEEK(0)
40 PRINT"CONTENUTO BYTE 65535: ";PEEK(65535)
50 STOP
```

POKE Tipo: Trattamento interno dati

Formato: POKE indirizzo, valore

dove: indirizzo è l'indirizzo di un byte;
valore è un numero tra 0 e 255.

Serve per scrivere un numero in un byte. Se non si rispettano i limiti per i due parametri si ha il messaggio: ?ILLEGAL QUANTITY.

E' una istruzione da usare con cautela, può produrre risultati indesiderati. E' molto utile quando si programma in Assembler, ma si può usare anche con succcesso in Basic. Nel programma esempio che segue, si scrivono le 26 lettere dell'alfabeto sulla prima riga del video e si colorano alternativamente in porpora e verde, andando a scrivere i D/CODE nelle posizioni della mappa video e i codici colore nelle corrispondenti posizioni della mappa colori.

```
1 REM ES41 PROVA POKE
10 MC=55296:MV=1024:PRINT"[redacted]";
20 FORK=1TO26:POKEMV+K-1,K:NEXTK
30 FORK=1TO26STEP2:POKEMC+K-1,4:POKEMC+K,5
35 NEXTK:STOP
```

POS Tipo: funzione lettura memoria

Formato: POS (argomento)

dove argomento è puramente formale e non ha effetto, puoi usare sempre zero.

Fornisce la posizione del cursore sul video, in senso orizzontale; ma considerando una linea video come formata da due righe, 40 + 40 caratteri, dà quindi come risultato un numero compreso tra 0 e 79.
Segue un semplice programma che ne illustra il significato.

```
1 REM ES42 PROVA POS
10 PRINT"[redacted]ABCD[redacted]EFGH[redacted]IJKL[redacted]";
20 PRINT"POS";POS(0)
30 INPUT"SCRIVI UNA FRASE: ";A$
40 PRINT"[redacted]";A$;" POS";POS(0)
50 STOP
```

PRINT Tipo: Operazione I/O dati

Formato: PRINT lista

dove lista può comprendere una e/o più variabili e/o costanti, separate tra loro da caratteri separatori e/o funzioni che influiscono sulla stampa e/o caratteri di controllo specifici.

Questa istruzione è principalmente diretta al video, ma il comando CMD può trasferire il suo effetto a altre periferiche, come abbiamo già visto.

Della PRINT diretta alla stampante dopo un comando CMD parliamo diffusamente nel Capitolo 2 del secondo Volume.

Elenchiamo qui alcune delle caratteristiche dell'istruzione, rimandando per maggior approfondimenti al Capitolo 3.

Se nella lista compare più di un elemento da stampare, si deve usare come carattere separatore il punto e virgola o la virgola o lo spazio, considerando le regole che seguono:

.. la virgola si può usare per separare due elementi qualsiasi, essa fa spaziare alla prossima zona di stampa; ogni zona di stampa sul video è di 10 caratteri, 4 zone per riga;
.. il punto e virgola si può usare per separare due elementi qualsiasi, esso non provoca spaziature tra gli elementi;
.. lo spazio si può usare solo tra costanti stringa o numeriche, non può essere usato quando uno degli elementi è una variabile numerica, cioè in tutti quei casi in cui l'interpretazione può essere ambigua.

Ogni elemento viene stampato con il suo formato, che per le stringhe è la stringa stessa, mentre per i numeri si ha uno spazio o il segno meno prima del numero e uno spazio dopo. Un numero di una cifra occupa 3 posizioni.
Nella lista di stampa le variabili stringa o le costanti stringa possono comparire una dopo l'altra senza separatori, cioè può essere omesso il punto e virgola o lo spazio.
La stampa comincia sempre dove si trova il cursore sul video. La posizione del cursore può essere modificata usando caratteri di controllo all'interno della lista.
Se la lista di stampa termina senza punteggiatura il cursore si posiziona, dopo la stampa, sulla linea seguente. Se invece la lista di stampa termina con virgola o punto e virgola, essi agiscono secondo le loro regole e quindi può anche succcedere che il cursore resti nella stessa riga dopo aver stampato tutta la lista. La punteggiatura a fine lista inibisce l'invio dei due caratteri ritorno-carrello e alimentazione-linea che sono altrimenti inviati.
Una istruzione PRINT senza lista fa stampare una linea bianca.
Durante la stampa di una lista, se i caratteri sono più di 40, la stampa prosegue sulla linea seguente.

Negli esempi di programmi esaminati fino ad ora abbiamo usato spesso il comando PRINT; per questa ragione non poniamo qui altri esempi e rimandiamo al Capitolo 3.

PRINT# Tipo: Operazione I/O dati

Formato: PRINT# lfn, lista

dove: lfn è il numero logico del file usato nella OPEN;
lista è una lista di variabili.

Questa operazione serve per scrivere su un file associato a una periferica. Per la periferica stampante rimandiamo al Capitolo 5. Per le periferiche nastro e disco l'argomento viene approfondito nel volume ad esse dedicato.
Qui ci limitiamo ad osservare che nella lista delle variabili si deve aver cura di forzare dei separatori tra i dati, in modo che essi vengano registrati sul supporto magnetico come caratteri separatori, e come tali riconosciuti in fase di lettura dati con INPUT# . Infatti i normali separatori punto e virgola e virgola o spazio tra le variabili della lista, agiscono a livello di aggiunta di caratteri spazio ai caratteri dei dati, ma non producono separatori riconoscibili in fase di lettura. Per produrre separatori riconoscibili, tra gli elementi della lista di stampa devono comparire in forma di stringa il RETURN (CHR$(13)), la virgola (",” o CHR$(44)).

READ Tipo: Trattamento interno dati

Formato: READ var [,var]....

dove var sono nomi di variabili, separati da virgole se più di uno.

Questa istruzione consente di trasferire ordinatamente i dati forniti dalle istruzioni DATA presenti nel programma in variabili. I nomi delle variabili devono concordare con il tipo dei dati che vengono via via prelevati. In caso di errore si ha il messaggio: ?SYNTAX ERROR AT LINE N, dove N è il numero della linea del DATA coinvolto e non della istruzione READ. Qualora si cerchi di prelevare più dati di quelli immagazzinati, si ha il messaggio: ?OUT OF DATA.

L'uso di READ e DATA consente di attingere rapidamente a costanti trasferendole in variabili, senza dover ricorrere all'istruzione LET, e lavorando solo all'interno del programma. Inoltre il comando RESTORE rende di nuovo disponibili i dati dall'inizio e può essere usato sia quando la lista di dati è esaurita che in qualunque altro momento.

Segue un esempio di uso.

```
1 REM ES45 PROVA READ
10 DIMM$(12):S$="            "
20 FORK=1TO12
30 READM$(K):M$(K)=LEFT$(M$(K)+S$,9)
40 NEXTK
50 FORK=1TO12:PRINTM$(K);" ";:NEXTK
```

```
60 PRINT:STOP
100 DATAGENNAIO,FEBBRAIO,MARZO,APRILE
101 DATAMAGGIO, GIUGNO,LUGLIO,AGOSTO
102 DATASETTEMBRE,OTTOBRE,NOVEMBRE,DICEMBRE
```

Come vedi abbiamo creato un magazzino di dati con i nomi dei mesi, usando 3 DATA, poi li abbiamo trasferiti nella variabile con indice M$, usando l'indice partendo da uno invece che da zero per avere la usuale corrispondenza, e abbiamo aggiunto degli spazi a destra ai nomi lunghi meno di nove caratteri.

REM Tipo: Servizio

Formato: REM annotazioni

Serve per introdurre commenti nei programmi e facilitarne la lettura. Dopo la parola chiave REM si può scrivere quello che si vuole usando qualunque carattere e anche le parole chiave del Basic.

Segue un programma esempio:

```
1 REM ES46 PROVA REM
10 REM PROGRAMMA PROVA
20 PRINT"TITOLO PROGRAMMA":PRINT
30 INPUT"NOME: ";N$:REM CHIEDE NOME
40 REM STAMPA:PRINT"NOME: ";N$
50 PRINT"STOP"
60 REM ****************************
70 REM    PROGRAMMA CALCOLO
80 REM ****************************
90 REM
95 ...........
```

Come vedi dalla linea 60 alla 90 c'è un tentativo di impaginazione estetica del programma. L'istruzione PRINT della linea 40 non verrà MAI eseguita, infatti al primo REM incontrato in una linea di programma essa viene saltata tutta nell'esecuzione. La linea 30 fa invece un uso corretto della REM ponendola dopo l'istruzione da eseguire.
Anche nelle frasi REM non si devono superare 2 righe video, massimo 80 caratteri. Si può, ma non ha molto senso, usarla in modo immediato.

RESTORE Tipo: Trattamento interno dati

Formato: RESTORE

Questa istruzione serve per riportare il puntatore interno all'inizio della lista dei dati che sono incorporati nel programma per effetto delle istruzioni DATA. Riportare il puntatore all'inizio significa rendere nuovamente disponibili dall'inizio tutti i dati e quindi poter usare di nuovo l'istruzione READ. Nell'esempio che segue, con la DATA sono disponibili 5 costanti stringa; esse vengono stampate sul video, senza la RESTORE della linea 50 non si potrebbe eseguire GOTO10, si avrebbe infatti il messaggio: ?OUT OF DATA.
Vai a vedere nel Capitolo 1 dell'ottavo Volume il programma RESTORE, nel quale ti mostriamo come ottenere prestazioni superiori a quelle del comando RESTORE disponibile in questa implementazione del Basic.

```
1 REM ES47 PROVA RESTORE
2 M$="■■■■■■■■■■■■■■■■■■■■■■■■■■■■ANCORA (S/N): "
5 DIMD$(5):OPEN2,0
6 REM APERTURA TASTIERA PER INPUT
7 REM SI EVITA ? IN INPUT
10 PRINT"■■   ■ELENCO ARGOMENTI■■"
20 FORK=1TO5:READA$:PRINT"    "A$" ";
25 INPUT#2,D$(K):PRINT
30 NEXTK
40 GOSUB300
45 GOSUB100:IFR$="N"THENSTOP
50 RESTORE:GOTO10
90 DATAPRIMO, SECONDO, TERZO
91 DATAQUARTO,QUINTO
100 PRINTM$;:INPUTR$
105 IFR$<>"S"ANDR$<>"N"THEN100
110 RETURN
300 PRINT"■ELABORAZIONE ARGOMENTI"
301 RETURN
```

RETURN Tipo: Controllo

Formato: RETURN

Questa istruzione (da non confondersi con il tasto RETURN) provoca l'uscita da un sottoprogramma e il ritorno al programma che ha eseguito il GOSUB. Essa va a

prelevare dall'area STACK l'ultimo indirizzo, quello affiorante, e lo usa per tornare alla linea di programma da dove proseguire. Se viene incontrato un RETURN in un programma si ha un messaggio di errore: ?RETURN WITHOUT GOSUB. Segue un semplice programma esempio:

```
1 REM ES48 PROVA RETURN
10 PRINT"VADO A SUBROUTINE":GOSUB100
20 PRINT"SONO TORNATO"
30 REM ERRATO NON STA IN UNA SUBROUTINE
31 RETURN
40 PRINT"QUI NON ARRIVO"
100 PRINT"SONO IN UNA SUBROUTINE":RETURN
```

RIGHT$ Tipo: Funzione stringa

Formato: RIGHT$ (stringa, numero)

dove:
stringa è una stringa, o una variabile stringa, o la somma di stringhe e/o variabili stringa;
numero è una espressione numerica compresa tra 0 e 255; se non intera i decimali vengono ignorati.

Fornisce il numero di caratteri richiesto, prelevandolo a destra della stringa. Se sono chiesti zero caratteri fornisce la stringa nulla. Se sono chiesti più caratteri di quelli che formano la stringa, essa viene fornita per intero.

Segue un semplice esempio.

```
1 REM ES49 PROVA RIGHT$
5 M$="STRINGA SENZA LA PRIMA META': "
10 INPUT"STRINGA: ";A$
20 PRINT"STRINGA INTERA: ";A$
30 PRINT"ULTIMI 5 CARATTERI: ";RIGHT$(A$,5)
40 PRINTM$;RIGHT$(A$,LEN(A$)/2)
50 STOP
```

RND Tipo: Funzione matematica

Formato: RND (argomento)

dove argomento è un numero.

Fornisce un numero pseudo-random prelevandolo dalla sequenza generata dal calcolatore mediante un algoritmo pseudo-casuale.
La sequenza dei numeri viene generata partendo da un numero iniziale che viene chiamato seme (seed). Il seme viene inizializzato al momento dell'accensione del calcolatore.
L'argomento agisce in questo modo:
.. se esso è negativo, predispone in base al suo valore il punto di partenza della sequenza dei numeri casuali, da cui andare a prelevare i numeri a caso, cioè inizializza la sequenza;
.. se esso è positivo non interessa il suo valore dato che fa solo proseguire nella sequenza; a parità del punto di partenza fornisce la stessa sequenza di numeri casuali;
.. se esso è zero il punto di partenza della sequenza viene influenzato dal valore del clock interno del calcolatore.

I numeri generati sono compresi tra 0 e 1 (minori di 1) e sono decimali. Per ottenere numeri interi in un dato intervallo si deve applicare un algoritmo.

Segue un programma esempio:

```
1 REM ES50 PROVA RND
5 OPEN4,4:CMD4
10 REM POSIZIONA INIZIO SEQUENZA
11 REM CON ARGOMENTO NEGATIVO RND
20 FORK=-5TO-1STEP1:PRINT
25 PRINT "INIZIO SEQUENZA CON RND(";K;")"
30 PRINTRND(K):GOSUB100
40 GOSUB400
45 GOSUB150
46 X=RND(0):PRINT"RND(0)=";X
47 X=RND(1):PRINT"RND(1)=";X
48 PRINT"RIPRISTINO BYTE 139/143"
50 GOSUB160:GOSUB400
55 NEXTK
60 PRINT#4:CLOSE4:STOP
100 FORL=0TO4:X(L)=PEEK(139+L):NEXTL
101 RETURN
150 FORL=0TO4:PRINT"BYTE(";139+L;")=";X(L)
151 NEXTL:RETURN
160 FORL=0TO4:POKE(139+L),X(L):NEXTL
161 RETURN
400 FORI=1TO2:FORJ=1TO3:PRINTRND(I*J);" ";
401 NEXTJ:PRINT:NEXTI:RETURN
```

In esso, che manda i risultati su stampante per poterli esaminare più comodamente, facciamo variare un numero K tra -5 e -1, e, usandolo come argomento di RND, posizioniamo ogni volta il seme a un valore determinato. Poi vengono generati 6 numeri a caso con argomento positivo. Ogni volta che si fa girare il programma i numeri risultano uguali, infatti si ha sempre il posizionamento con lo stesso argomento negativo. Dopo aver posizionato il seme, con K negativo, memorizziamo in 5 variabili X(L) il contenuto dei 5 byte coinvolti dal sistema nella operazione RND (da 139 a 143). Poi stampiamo RND(0) e RND(1), che risultano ogni ciclo diversi. Alla linea 50, con il sottoprogramma 160, ripristiniamo i valori dei 5 byte da 139 a 143, con i valori che avevano al momento della RND(K) con K negativo, e stampiamo di nuovo 6 numeri casuali con argomento positivo. Come puoi vedere le due sequenze di numeri sono uguali ogni volta.

RISULTATI PROGRAMMA ES50

```
INIZIO SEQUENZA CON RND(-5 )
 3.73720468E-08
 .898884767    .930769958    .534739951
 .690061376    .718249081    3.80238951E-03
BYTE( 139 )= 104
BYTE( 140 )= 32
BYTE( 141 )= 131
BYTE( 142 )= 0
BYTE( 143 )= 0
RND(0)= .285157919
RND(1)= .458660293
RIPRISTINO BYTE 139/143
 .898884767    .930769958    .534739951
 .690061376    .718249081    3.80238951E-03

INIZIO SEQUENZA CON RND(-4 )
 2.99214662E-08
 .402343613    .92613015    .7171307
 .137023837    .770137118    .319573971
BYTE( 139 )= 104
BYTE( 140 )= 0
BYTE( 141 )= 131
BYTE( 142 )= 0
BYTE( 143 )= 0
RND(0)= .230470896
RND(1)= .431733826
```

```
RIPRISTINO BYTE 139/143
 .402343613   .92613015   .7171307
 .137023837   .770137118   .319573971

INIZIO SEQUENZA CON RND(-3 )
 4.48217179E-08
 .931279155   .556729296   .57212578
 .0406100041   .181847568   .172528894
BYTE( 139 )= 104
BYTE( 140 )= 64
BYTE( 141 )= 130
BYTE( 142 )= 0
BYTE( 143 )= 0
RND(0)= .148440719
RND(1)= .763706377
RIPRISTINO BYTE 139/143
 .931279155   .556729296   .57212578
 .0406100041   .181847568   .172528894

INIZIO SEQUENZA CON RND(-2 )
 2.99205567E-08
 .865554613   .846128798   .981100118
 .986081582   .514064711   .768773897
BYTE( 139 )= 104
BYTE( 140 )= 0
BYTE( 141 )= 130
BYTE( 142 )= 0
BYTE( 143 )= 0
RND(0)= .519533396
RND(1)= .747457131
RIPRISTINO BYTE 139/143
 .865554613   .846128798   .981100118
 .986081582   .514064711   .768773897

INIZIO SEQUENZA CON RND(-1 )
 2.99196472E-08
 .328780872   .978964086   .895758909
 .161031701   .0224078245   .0365292135
BYTE( 139 )= 104
BYTE( 140 )= 0
BYTE( 141 )= 129
BYTE( 142 )= 0
BYTE( 143 )= 0
```

```
RND(0)= 7.81309605E-03
RND(1)= .336002869
RIPRISTINO BYTE 139/143
 .328780872    .978964086    .895758909
 .161031701    .0224078245    .0365292135
```

Segue un semplice programma che lancia 20 volte un dado. Osserva alla linea 20 come si opera affinchè il numero sia compreso tra 1 e 6.

```
1 REM ES50BIS LANCIO DADI
5 PRINT" LANCIO 20 VOLTE UN DADO"
10 FORK=1TO4:FORJ=1TO5
20 X%=1+6*RND(K*J)
30 PRINTX%;" ";:NEXTJ:PRINT
40 NEXTK
50 STOP
```

RUN Tipo: Controllo

Formato: RUN [numero-linea]

dove numero-linea, se presente, deve essere un numero di linea esistente nel programma.

Questo comando è usato in modo immediato per far partire i programmi. Se manca il numero-linea, il programma parte dalla prima linea presente. Se numero-linea è presente il programma parte da quella linea. Prima di far partire il programma viene eseguito un CLR implicito di tutte le variabili.
Esso può essere usato anche dall'interno di un programma, che in questo caso fa ripartire se stesso, dalla linea citata o dall'inizio.

SAVE Tipo: Operazione I/O programmi

Formato: SAVE [fn] [,dn] [,ind]

dove i parametri hanno il solito significato precisato all'inizio di questo paragrafo.

Serve per memorizzare programmi su nastro o su disco. Di norma si usa il nome di un file per memorizzare il programma, questo per poterlo richiamare poi con un nome; comunque il nome può anche essere tralasciato quando si memorizza su nastro, deve essere presente se si usa il disco. Lo fn può essere una variabile stringa o

una stringa. Il sistema attribuisce al file il tipo PRG, che significa PROGRAMMA. Se viene omesso dn, esso è assunto uguale a 1 e quindi la memorizzazione avviene su nastro. Per il disco si deve porre dn=8. Dopo il salvataggio del programma esso è ancora in memoria e può continuare ad operare. Se il comando è dato dall'interno di un programma, esso salva se stesso e poi prosegue con l'istruzione successiva. Il terzo parametro, ind, può essere uguale a 1 per memorizzare il programma, mantenendo, al momento del LOAD, invariato il punto di inizio in memoria (diverso dal normale 2049). Per il nastro si può usare ind=2 per far memorizzare una segnalazione di fine-nastro dopo il programma; ind=3 per ottenere i due effetti combinati, cioè indirizzo di caricamento invariato e segnalazione di fine-nastro.

Esempi del comando:

SAVE	memorizza su nastro senza nome
SAVE "PRO1"	memorizza su nastro con nome PRO1
SAVE "PRO1",8	memorizza su disco con nome PRO1
SAVE "PRO1",1,1	memorizza su nastro mantenendo l'indirizzo di inizio
SAVE "PRO1",1,2	memorizza su nastro con segnalazione di fine-nastro
SAVE "PRO1",1,3	memorizza mantenendo l'indirizzo di inizio e con segnalazione di fine-nastro
SAVE "PRO1",8,1	memorizza su disco mantenendo l'indirizzo di inizio.

SGN Tipo: Funzione matematica

Formato: SGN (espressione)

dove espressione deve essere numerica.

Fornisce un risultato intero:

.. 1 se il valore dell'argomento è positivo,
.. 0 se il valore dell'argomento è zero,
..-1 se il valore dell'argomento è negativo.

Segue un semplice esempio:

```
1 REM ES53 PROVA SGN
10 INPUT"NUMERO: ";N
20 PRINT"SGN(";N;")=";SGN(N);
30 ON(SGN(N)+2)GOTO40,50,60
40 PRINT" NEGATIVO":STOP
50 PRINT" NULLO":STOP
60 PRINT" POSITIVO":STOP
```

SIN Tipo: Funzione matematica

Formato: SIN (numero)

dove numero può anche essere una variabile numerica o un'espressione numerica.

L'argomento deve essere espresso in radianti. Segue un semplice esempio:

```
1 REM ES54 PROVA SIN
10 INPUT"ARGOMENTO IN RADIANTI: ";N
20 PRINT"SIN(";N;")=";SIN(N)
30 PRINT"COS(";N;")=";SIN(N+π/2)
40 STOP
```

SPC Tipo: Funzione stampa

Formato: SPC (espressione)

dove espressione deve dare un valore numerico compreso tra 0 e 255 per stampante e nastro, tra 0 e 254 per il disco.

Se il valore di espressione non è intero, i decimali vengono troncati. La funzione provoca il salto di N spazi, se N è il valore dell'espressione; quando provoca uno spazio nell'ultima posizione della linea si ha un passaggio automatico alla linea successiva.

Segue un esempio di uso sul video.

```
1 REM ES55 PROVA SPC
10 A$="PRIMA STRINGA"
15 B$="SECONDA STRINGA"
20 PRINTA$;SPC(7);B$
```

```
25 PRINTA$SPC(7)B$
30 A1$="BELLO":A2$="BELLISSIMO"
35 A3$="PIACEVOLE":A4$="STUPENDO"
40 PRINT"▒  "A1$SPC(20-LEN(A1$))A2$
45 PRINT"  "A3$SPC(20-LEN(A3$))A4$
50 STOP
```

Nota l'uso sofisticato di SPC, usando nell'argomento la lunghezza della stringa appena stampata, per ottenere gli incolonnamenti. Segue un risultato su stampante del programma, ottenuto scrivendo in immediato:

OPEN4,4:CMD4:GOTO1

come puoi notare il carattere di controllo cursore-giù incorporato nella stringa di inizio della linea 40 ha sulla stampante l'effetto di cambiare il set di caratteri.

RISULTATI SU STAMPANTE DI ES55

```
PRIMA STRINGA          SECONDA STRINGA
PRIMA STRINGA          SECONDA STRINGA
  bello                  bellissimo
  Piacevole              stupendo


break in 50
```

SQR Tipo: Funzione matematica

Formato: SQR (espressione)

dove espressione deve essere un'espressione numerica con valore positivo.

Fornisce la radice quadrata dell'argomento.

Segue un esempio.

```
1 REM ES56 PROVA SQR
10 INPUT"NUMERO: ";N
20 IFN<0THENPRINT"NUMERO NEGATIVO":GOTO10
30 PRINT"SQR(";N;")=";SQR(N)
40 STOP
```

STATUS Tipo: Funzione lettura memoria

Formato: STATUS

Fornisce un valore numerico intero che informa sull'esito dell'ultima operazione di I/O eseguita su un file. Si può usare per qualunque periferica sulla quale si è aperto un file. Si abbrevia in ST.
I valori di ST hanno il seguente significato:

Pos.bit	Valore decim.	Lett. nastro	Verify/Load nastro	Bus ser. I/O
0	1			fuori tempo scritt.
1	2			fuori tempo lett.
2	4	blocco corto		
3	8	blocco lungo		
4	16	err.lett.	errore	
5	32	errore checksum		
6	64	fine file		EOI
7	—128	fine nastro		device non presente

Quando ST è zero significa che l'operazione è andata bene, se diverso da zero, si deve controllare il valore per stabilire cosa fare. Il valore 64 e il valore -128 non sono errori, ma solo utili indicazioni per il programma. Devi fare attenzione e memorizzare ST in una variabile di comodo, se lo analizzi in un passo di programma successivo, infatti ogni operazione file interviene su ST. Tipico esempio è la lista di un file sequenziale letto da nastro; quando leggi l'ultimo dato, ST diventa 64, se non lo memorizzi e vai a stampare (la stampante è aperta come file), ST cambia, quando vai ad analizzare la segnalazione per vedere se è finito il file non trovi più il valore 64.

STEP Tipo: Controllo

Formato: STEP espressione

dove espressione deve essere numerica, positiva o negativa.

Può essere presente in una frase FOR dopo il valore finale della variabile di controllo del ciclo. Viene usato ogni ciclo per modificare il contatore del ciclo. Non ha senso porlo uguale a zero, perchè produce un ciclo infinito.
Vai a rivedere la descrizione di FOR.

STOP Tipo: Controllo

Formato: STOP

Ferma il programma con lo stesso effetto di quando si preme il tasto RUN/STOP. Compare il messaggio: ?BREAK IN LINE .... Per far proseguire il programma si deve usare il comando CONT. E' molto utile per verificare particolari risultati durante le prove di un programma; dopo gli STOP possono essere eliminati.

STR$ Tipo: Funzione stringa

Formato: STR$ (espressione)

dove espressione deve essere numerica.

Trasforma il valore numerico di espressione in stringa. Accetta come argomento un numero intero o decimale con segno.

Segue un esempio:

```
1 REM ES60 PROVA STR$
5 M$="LUNGHEZZA STRINGA GENERATA:  "
10 INPUT"NUMERO:  ";N
20 PRINT"STR$(";N;")=";STR$(N);"*"
25 PRINTM$;LEN(STR$(N))
27 PRINTN;"*"
29 PRINTSTR$(N);"*"
30 STOP
```

RISULTATI ES60

```
NUMERO:
STR$( 5678 )=-5678*
LUNGHEZZA STRINGA GENERATA:  5
 5678 *
 5678*


STR$(-34567878 )=-34567878*
LUNGHEZZA STRINGA GENERATA:  9
-34567878 *
-34567878*
```

```
STR$( 4.5678E-09 )= 4.5678E-09*
LUNGHEZZA STRINGA GENERATA:  11
 4.5678E-09 *
 4.5678E-09*
```

Come puoi vedere dai risultati dell'esempio la stringa ottenuta dalla funzione ha un carattere meno del numero in formato stampa, manca lo spazio dopo il numero; abbiamo usato l'artificio di stampare un asterisco a chiusura della stampa. Come vedi si possono scrivere i numeri in qualunque formato.

SYS Tipo: Controllo

Formato: SYS indirizzo

dove indirizzo è un indirizzo di memoria.

Questo comando serve per andare ad eseguire un programma scritto in linguaggio macchina e memorizzato a partire dall'indirizzo citato. Indirizzo può anche essere un'espressione numerica, purchè fornisca un numero valido, tra 0 e 65535. Il programma in linguaggio maccchina deve terminare con il comando RTS, che fa proseguire il programma Basic dall'istruzione seguente la SYS. Il codice decimale di RTS è 96, puoi provare a scrivere:

POKE4400,96:SYS4400

fa andare a 4400, dove trova RTS e quindi torna al Basic.

Il comando SYS 64738 ripristina le condizioni iniziali del calcolatore al momento dell'accensione, mandando in esecuzione una routine del sistema operativo.

TAB Tipo: Funzione stampa

Formato: TAB (espressione)

dove espressione deve fornire un valore numerico compreso tra 0 e 255.

Il valore di espressione fornisce il numero di posizioni di stampa da contare partendo dall'inizio della linea attuale per arrivare alla posizione di stampa quando è usato in una istruzione PRINT diretta al video. In questo caso, se l'argomento di TAB dà un valore inferiore alla posizione attuale, del cursore, essa non agisce. Quando invece TAB compare in una istruzione di PRINT diretta alla stampante, l'argomento rappresenta il numero di spazi da lasciare prima della prossima stampa, partendo dalla posizione attuale, cioè si comporta come SPC. Segue un programma esempio, che abbiamo ricavato modificando l'esempio riportato per SPC.

```
1 REM ES62 PROVA TAB
10 A$="PRIMA STRINGA"
15 B$="SECONDA STRINGA"
20 PRINTA$;TAB(15);B$
25 PRINTA$TAB(15)B$
30 A1$="BELLO":A2$="BELLISSIMO"
35 A3$="PIACEVOLE":A4$="STUPENDO"
40 PRINT"▒  "A1$TAB(20)A2$
45 PRINT"  "A3$TAB(20)A4$
50 STOP
```

TAN Tipo: Funzione matematica

Formato: TAN (espressione)

dove espressione deve dare un valore numerico in radianti dell'angolo di cui si vuole la tangente.

Segue un esempio:

```
1 REM ES63 PROVA TAN
10 INPUT"ARGOMENTO IN RADIANTI: ";N
20 PRINT"TAN(";N;")=";TAN(N)
50 STOP
```

THEN Tipo: Controllo

Formato: THEN numero linea
THEN istruzione

è collegata alla parola chiave IF alla quale rimandiamo.

TIME Tipo: Funzione lettura memoria

Formato: TI

fornisce il contenuto del clock interno in sessantesimi di secondo. Questo numero diviso 60 fornisce i secondi trascorsi dall'accensione del calcolatore, o dal ripristino delle condizioni iniziali, o dall'azzeramento di TI, che però non può essere ottenuto direttamente, ma azzerando la stringa TI$.

Segue un esempio, che tratta sia TI che TI$.

```
1 REM ES65 PROVA TIME
10 PRINT"SEC. DALL'ACCENSIONE ";
15 PRINTTI/60
20 PRINT"TI$=";TI$
30 TI$="000000"
35 PRINT"SEC. DOPO AZZERAMENTO TI$ ";
40 PRINTTI/60
60 REM CALCOLO INTERVALLO TEMPO
61 FORI=1TO4
70 INPUT"LIMITE CICLO FOR: ";N
72 A=TI
75 FORK=0TON:NEXTK
77 B=TI
79 PRINT"INTERVALLO: ";(B-A)/60
80 NEXTI:STOP
```

Come puoi vedere, viene inizialmente stampato il contenuto di TI/60 e di TI$. La stringa contenuta in TI$ va letta: hhmmss, cioè ore, minuti, secondi.
Alla linea 30 viene azzerato TI$ e poi viene stampato di nuovo il contenuto di TI/60. Poi, per 4 volte, viene chiesto un limite N per il ciclo FOR e viene calcolato l'intervallo di tempo che trascorre per eseguire FOR...NEXT N volte.

TIME$ Tipo: Funzione lettura/scrittura memoria

Formato TI$

Vedi l'esempio precedente.

TO Tipo: Controllo

Formato: TO limite

Fa parte del ciclo FOR; rimandiamo alle spiegazioni relative.

USR Tipo: Funzione controllo

Formato: USR (espressione)

dove espressione deve essere numerica.

Serve per andare ad eseguire un programma in linguaggio macchina, il cui indirizzo deve essere stato preventivamente memorizzato nei byte 785 e 786, con delle

istruzioni POKE. Il valore di espressione viene memorizzato nell'accumulatore floating-point che inizia al byte 97; in tale accumulatore si trova poi il risultato del calcolo eseguito. Il risultato comunque viene reso disponibile al programma Basic, come valore della funzione. Il programma in linguaggio macchina deve terminare con l'istruzione RTS.

VAL Tipo: Funzione conversione

Formato: VAL (stringa numerica)

dove stringa numerica deve essere un qualunque numero scritto come stringa.

La funzione trasforma il numero in stringa. Se la stringa ha il primo carattere diverso da spazio che non è un segno o una cifra, il valore fornito è zero. La conversione si ferma al primo carattere non numerico incontrato, senza segnalare errore.

Segue un esempio:

```
1 REM ES69 PROVA VAL
10 INPUT"STRINGA NUMERICA: ";N$
20 PRINT"VAL(";N$;")=";VAL(N$)
30 STOP
```

VERIFY Tipo: Operazione I/O programmi

Formato: VERIFY [fn] [,dn]

dove i parametri hanno il solito significato.

Può essere usata in modo immediato o da programma. Esegue la verifica di un file programma confrontandolo con quello presente in memoria. E' buona norma verificare sempre le registrazioni effettuate. Se si omette dn, il sistema assume si tratti del nastro. Sempre nel caso del nastro se si omette il nome, viene verificato il primo programma incontrato; il nastro deve essere ovviamente correttamente posizionato. Per il disco è obbligatorio il nome del file.
Il nome del programma può essere una stringa tra virgolette o una variabile, o una espressione stringa.
Durante la verifica si svolge un colloquio tramite video. Nel caso del nastro viene chiesto di premere il tasto PLAY di avviamento.

WAIT Tipo: Controllo

Formato: WAIT indirizzo, maschera-1, [maschera-2]

dove: indirizzo è un indirizzo di memoria, cioè un byte che si desidera analizzare; maschera-1, deve essere un numero compreso tra 0 e 255, viene usato per operare un AND con il risultato dell'operazione di OR-esclusivo (XOR) tra il contenuto di indirizzo e maschera-2. Questa maschera estrae dal risultato i bit 1 corrispondenti ai suoi bit 1;
maschera-2, deve essere un numero compreso tra 0 e 255, viene usato per operare un OR-esclusivo con il contenuto dell'indirizzo; esso fornisce bit 0 dove trova bit uguali e bit 1 dove trova bit diversi.
Se maschera-2 manca, esso viene assunto uguale a zero.

Le regole dell'operazione di OR-esclusivo sono le seguenti:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Tieni presente che con WAIT puoi entrare in un ciclo infinito. Segue un programma che puoi provare; in caso per uscire puoi tenere premuto il tasto RUN/STOP e premere RESTORE.

```
1 REM ES71 PROVA WAIT
10 INPUT"PARAMETRI WAIT: ";X,Y,Z
15 A=TI
16 WAITX,Y,Z
17 B=TI
20 PRINT"WAIT TERMINATO"
25 PRINT"INTERVALLO SEC.: ";(B-A)/60
30 STOP
```

Il programma prosegue se il risultato della operazione logica sul contenuto dell'indirizzo risulta vero, cioè diverso da zero, mentre torna all'operazione di WAIT se il risultato risulta falso, cioè uguale a zero.

Se provi con X=162, che è il byte basso del clock, e con diversi valori per Y e Z puoi ottenere alternativamente cicli infiniti o intervalli di riposo più o meno lunghi.

Puoi provare anche il programma ES71BIS, che segue, dove vengono chiesti,

l'indirizzo X di un byte e le due maschere, poi viene chiesto il valore da scrivere nel byte X e viene lanciata l'istruzione WAIT. Puoi usare per X l'indirizzo di un byte in RAM situato dopo il programma; se usi 4000 sei sicuro di andar bene. Puoi provare a scrivere in 4000 diversi valori mantenendo fisse le maschere. Puoi usare per X anche 162, che è il byte basso del clock e quindi si modifica da solo, e inizializzarlo con un numero scelto da te.

```
1 REM ES71BIS PROVA WAIT
10 INPUT"PARAMETRI WAIT: ";X,Y,Z
13 PRINT"VALORE PER BYTE";X;": ";:INPUTN
15 PRINT:POKEX,N:A=TI
16 WAITX,Y,Z
17 B=TI
20 PRINT"WAIT TERMINATO"
23 PRINT"CONTENUTO X: ";PEEK(X)
25 PRINT"INTERVALLO SEC.: ";(B-A)/60
30 STOP
```

I due programmi ti fanno vedere la durata approssimata del ciclo WAIT. Il programma ES71BIS stampa anche il contenuto di X dopo WAIT; esso sarà il contenuto iniziale se il byte non viene automaticamente modificato dal sistema in tempo reale, come il clock.

Esempi numerici delle operazioni XOR e AND:

1)

Cont.Ind.	Maschera-1	Maschera-2
3	3	255
00000011	00000011	11111111
XOR tra	00000011 e 11111111 dà	11111100
AND tra	11111100 e 00000011 dà	00000000

producendo un ciclo di attesa infinito.

2)

3	7	255
00000011	00000111	11111111
XOR tra	00000011 e	
	11111111 dà	11111100
AND tra	11111100 e	
	00000111 dà	00000100

e il ciclo termina subito.

3)

3	255	0
00000011	11111111	00000000
XOR tra	00000011 e	
	00000000 dà	00000011
AND tra	00000011 e	
	11111111 dà	00000011

e il ciclo termina subito.

CAPITOLO 3

COLLOQUIO VIDEO/TASTIERA

3.1 I DUE STATI DEL CALCOLATORE

Quando accendi il tuo COMMODORE 64 e compare la scritta iniziale è in funzione il programma EDITOR e il calcolatore si trova nello STATO SISTEMA.

I due stati possibili per il calcolatore sono:
.. STATO SISTEMA;
.. STATO PROGRAMMA.

Siamo nel primo stato quando comunichiamo con il sistema calcolatore impartendo comandi tramite la tastiera. Siamo nel secondo stato quando sta funzionando un programma in Basic caricato nella memoria RAM.
Quando il programma Basic si interrompe per uno dei seguenti motivi:
.. abbiamo premuto il tasto RUN/STOP;
.. il programma ha incontrato uno STOP o un END oppure ha eseguito l'ultima istruzione fisicamente presente;
.. il sistema ha interrotto il programma segnalando con un messaggio la presenza di un errore;
si passa dallo stato programma allo stato sistema. Quando vengono eseguite istruzioni Basic in modo immediato, si transita dallo stato programma per il tempo necessario all'esecuzione delle istruzioni, e si torna allo stato sistema.
Un programma Basic può far rimanere indefinitamente il calcolatore nello stato programma; per esempio se contiene un ciclo infinito, oppure se si mette in attesa della pressione di un particolare tasto (GET) e questo non viene premuto. Quando il programma Basic è in attesa di ricevere dati con una istruzione INPUT non viene sentito il tasto RUN/STOP; dopo vari tentativi si può riuscire ad interrompere il programma premendo contemporaneamente RUN/STOP e RESTORE.

3.2 USO TASTI E CARATTERI DI CONTROLLO

Nel Paragrafo 1.2 abbiamo visto il significato dei tasti che permettono di muoversi sul video; li riassumiamo qui. Nella descrizione che segue i termini "viene visualiz-

zato" o "appare" si riferiscono all'effetto dei tasti quando si sta scrivendo dopo avere "aperto le virgolette" (quote mode).

CLR/HOME

viene visualizzato come una S in campo inverso, manda il cursore nell'angolo in alto a sinistra del video, senza cancellare il suo contenuto. Corrisponde a CHR$(19).

SHIFT + CLR/HOME

viene visualizzato come un cuore in campo inverso, pulisce il video e manda il cursore nell'angolo in alto a sinistra. Corrisponde a CHR$(147).

CRSR/frecce-verticali

viene visualizzato come Q in campo inverso, sposta il cursore di una posizione verso il basso. Corrisponde a CHR$(17).

SHIFT + CRSR/frecce-verticali

viene visualizzato come un cerchio in campo inverso, sposta il cursore di una posizione verso l'alto. Corrisponde a CHR$(145).

CRSR/frecce-orizzontali

viene visualizzato come una parentesi quadra destra in campo inverso, sposta il cursore di una posizione verso destra. Corrisponde a CHR$(29).

SHIFT + CRSR/frecce-orizzontali

viene visualizzato come una barretta verticale in campo inverso, sposta il cursore di una posizione verso sinistra. Corrisponde a CHR$(157).

Per modificare i dati presenti sul quadro video si deve portare il cursore nella posizione desiderata e, poi, o riscrivere sopra o inserire o cancellare caratteri. I tasti per inserire o cancellare sono:

INST/DEL

cancella il carattere a sinistra del cursore e sposta il cursore e tutto quello che lo segue verso sinistra di una posizione. Corrisponde a CHR$(20).

SHIFT + INST/DEL

crea uno spazio a sinistra del cursore, spostando il cursore e tutto quello che lo segue di una posizione verso destra. Corrisponde a CHR$(148).

Per passare alla visualizzazione dei caratteri in campo inverso si devono usare contemporaneamente i due tasti CTRL e 9, che appare come una R in campo inverso e corrisponde a CHR$(18). Per tornare al modo normale si devono premere contemporaneamente i due tasti CTRL e 0, che appare come una barrettina orizzontale in campo inverso e corrisponde a CHR$(146).

Devi dedicare una particolare attenzione al video quando apri le virgolette per scrivere una stringa. I tasti di controllo possono essere usati all'interno di una stringa e appaiono con il carattere descritto. Quelli che creano problemi sono INST/DEL e SHIFT + INST/DEL, e quindi è meglio non usarli.
Se devi andare a modificare qualcosa all'interno di una stringa ti conviene uscire con il tasto RETURN, e poi posizionarti spostando il cursore all'interno della stringa e usare nel solito modo i tasti di cancelazione e inserzione.
I tasti di controllo, invece di comparire all'interno di una stringa, possono essere creati usando la funzione CHR$ avente come argomento il codice ASCII corrispondente.
Ricorda che nei listati su stampante dei programmi i caratteri di controllo contenuti all'interno delle stringhe possono apparire diversamente che sul video; questo dipende dalle caratteristiche della stampante.

Le caratteristiche dell'EDITOR consentono di scrivere facilmente i programmi in Basic. Sul video appaiono le linee che tu scrivi; se le scrivi in disordine basta che tu usi il comando LIST seguito dai possibili parametri per avere in ordine il listato delle parti di programma che ti interessano. Per correggere linee di programma puoi portarti con il cursore sulla posizione desiderata e correggere; quando premi RETURN la correzione viene accettata.
Puoi cambiare il numero di linea, però ricordati di abolire la linea precedente. Per abolire una linea di programma basta scrivere il suo numero e premere RETURN.
Puoi ripetere linee di programma, che sono poco diverse tra loro, andando a cambiare il numero di linea, poi puoi listarle e apportare nelle copie le necessarie modifiche.
Questa ricchezza di prestazioni presenta anche dei pericoli; infatti si possono produrre modifiche non desiderate. Dopo una modifica ti conviene sempre listare il pezzo di programma coinvolto e controllare.

3.3 CONTROLLO COLORE

I byte che controllano il colore quando si usa il calcolatore in modo normale (no grafica, no multicolor) sono i seguenti:

.. 53280 colore bordo (codice colore +240);
.. 53281 colore sfondo (codice colore +240);
.. 646 colore caratteri (codice colore).

Al momento dell'accensione del calcolatore la situazione è la seguente:

.. 53280 contiene 254 (codice colore 14 + 240);
.. 53281 contiene 246 (codice colore 6 + 240);
.. 646 contiene 14 (codice colore 14);

il colore del cursore e il colore del bordo sono uguali, azzurro, mentre il colore dello sfondo è blu.

Sotto EDITOR, se premi contemporaneamente il tasto CTRL e un tasto numerico da 1 a 8 oppure il tasto COMMODORE e un tasto numerico da 1 a 8, vedi cambiare il colore del cursore; quello che scrivi dopo appare nel colore del cursore.
Puoi incorporare in una stringa i caratteri speciali che corrispondono alla pressione contemporanea di queste coppie di tasti; quando la stringa viene stampata si ha lo stesso effetto di cambio colore.
Puoi scrivere con POKE nel byte 646 i codici ASCII che corrispondono a queste coppie di tasti e ottieni lo stesso effetto di cambio colore.
I codici colore, cioè i numeri corrispondenti ai colori, non corrispondono ai tasti numerici che si usano; si ha che:
..premendo i tasti numerici contemporaneamente a CTRL, al tasto 1 corrisponde il numero colore 0 e, andando in sequenza, al tasto 8 il numero colore 7;
..premendo i tasti numerici contemporaneamente a COMMODORE, al tasto 1 corrisponde il numero colore 8 e, andando in sequenza, al tasto 8 il numero colore 15.

Segue il programma COLORI1, nel quale puoi vedere tutte queste cose. Esso opera così:
.. pone nella stringa A$, di 16 posizioni, i caratteri che si generano premendo contemporaneamente CTRL e i tasti da 1 a 8 e COMMODORE e i tasti da 1 a 8;
.. prepara con dei DATA i nomi dei 16 colori;
.. con un FOR che esegue 16 cicli, preleva dalla stringa A$ i caratteri-colore in sequenza, li stampa uno per volta, ottenendo di modificare il colore del cursore e quindi dei caratteri, e va a leggere e memorizzare nella matrice C% di 16 righe e 3 colonne i contenuti dei byte 53280, 53281 e 646;

.. apre la stampante e stampa una tabellina, nella quale sono riportati i nomi dei colori, i contenuti dei tre byte citati, il carattere corrispondente al colore come appare tra le virgolette e il codice ASCII corrispondente (quello da usare come argomento della funzione CHR$).

```
1 REM COLORI1
5 DIMC%(15,2),C$(15):S$="                    "
6 M$="COLORE        53280 53281 646    CAR. CHR$"
10 A$="■■■■■■■■■■■■■■■■"
11 DATANERO,BIANCO,ROSSO, CIANO,PORPORA
12 DATAVERDE,BLU,GIALLO,ARANCIONE, MARRONE
13 DATAROSSO-CHIARO,GRIGIO-1,GRIGIO-2
14 DATAVERDE-CHIARO,AZZURRO,GRIGIO-3
15 FORK=0TO15
20 C$(K)=MID$(A$,K+1,1):PRINTC$(K);
25 C%(K,0)=PEEK(53280)-240:REM COLORE BORDO
30 C%(K,1)=PEEK(53281)-240:REM COLORE SFONDO
35 C%(K,2)=PEEK(646):REM COLORE CURSORE
40 NEXTK:PRINT
43 OPEN4,4:CMD4
45 PRINT"        CONTENUTI BYTE COLORI"
50 PRINT:PRINTM$:PRINT
55 FORK=0TO15:READC$:B$=LEFT$(C$+S$,13)
57 X$=STR$(C%(K,0)):X$=LEFT$(X$+S$,5)
58 Y$=STR$(C%(K,1)):Y$=LEFT$(Y$+S$,6)
60 PRINTB$;X$;Y$;
61 X$=STR$(C%(K,2)):X$=LEFT$(X$+S$,6)
63 PRINTX$;
65 PRINTCHR$(34)C$(K)CHR$(34);
67 PRINT" ";ASC(C$(K))
70 NEXTK:PRINT#4:CLOSE4
```

RISULTATI PROGRAMMA COLORI

CONTENUTI BYTE COLORI

COLORE	53280	53281	646	CAR.	CHR$
NERO	14	6	0	"■"	144
BIANCO	14	6	1	"■"	5
ROSSO	14	6	2	"■"	28
CIANO	14	6	3	"■"	159
PORPORA	14	6	4	"■"	156
VERDE	14	6	5	"■"	30
BLU	14	6	6	"■"	31
GIALLO	14	6	7	"■"	158
ARANCIONE	14	6	8	"■"	129
MARRONE	14	6	9	"■"	149
ROSSO-CHIARO	14	6	10	"■"	150
GRIGIO-1	14	6	11	"■"	151
GRIGIO-2	14	6	12	"■"	152
VERDE-CHIARO	14	6	13	"■"	153
AZZURRO	14	6	14	"■"	154
GRIGIO-3	14	6	15	"■"	155

Consultando la tabellina, trovi naturalmente che il contenuto dei byte 53280 e 53281 resta sempre uguale, infatti la PRINT del carattere-colore non modifica sfondo e bordo, ma solo il colore del cursore.

Abbiamo preparato il programma COLORI2, per provare le combinazioni di colore e trovare quelle che ci soddisfano.

```
1 REM COLORI2
6 PRINT"■■■
```

```
26 X$=D$(B%):Y$=D$(S%):Z$=D$(C%)
27 POKE53280,B%OR240
28 POKE53281,S%OR240
29 POKE646,C%
30 PRINT"■■BORDO: "X$;" (";B%;")"
35 PRINT"■■SFONDO: "Y$;" (";S%;")"
40 PRINT"CARATTERI: "Z$;" (";C%;")"
41 PRINT"BYTE 646= ";PEEK(646)
44 PRINT"BYTE 53281= ";PEEK(53281)
45 PRINT"BYTE 53280= ";PEEK(53280)
50 PRINT"■■PREMI UN TASTO PER CONTINUARE"
55 GETB$:IFB$=""THEN55
60 GOTO15
```

In esso ti viene chiesto il codice colore per bordo, sfondo e carattere (cursore), poi vengono modificati i contenuti dei tre byte relativi con delle istruzioni POKE e vengono stampati i nomi dei colori scelti, i numeri-colore corrispondenti e i contenuti dei tre byte. Alla linea 55 abbiamo messo un ciclo di attesa, per interromperlo devi premere un qualunque tasto e proseguire.
Vedrai che se tra una prova e l'altra usi come colore dello sfondo il codice che hai usato precedentemente per i caratteri le 3 prime righe di stampa scompaiono dal video, ànche se esistono ancora, infatti se il colore dello sfondo e quello dei caratteri coincidono non si può leggere.
Vedrai anche che queste prime 3 righe dopo i cambi dei colori rimangono di un colore diverso, infatti nella mappa colori del video il sistema pone il colore nel momento che scrive un carattere e non lo modifica se cambia il colore del cursore. Se non fosse così non si potrebbero ottenere quadri video con tanti colori diversi.

Da quanto abbiamo visto, il sistema dispone di 16 colori, essi possono essere usati sia per il bordo, che per lo sfondo, che per i caratteri.
Ne' volume dedicato alla grafica approfondiremo anche l'argomento colore.

Per poter vedere tutti insieme i 16 colori disponibili abbiamo preparato il programma COLORI3; in esso vengono stampati sul video 16 rettangoli ognuno di un colore diverso. Quando il colore del carattere uguaglia il colore dello sfondo non si vede il carattere.

```
1 REM COLORI3
3 DIMC$(15):B$="●":D$="":PRINT"■";
4 FORK=0TO9:D$=D$+B$:NEXTK
5 A$="[illegible]":POKE53281,241
6 FORK=0TO15:C$(K)=MID$(A$,K+1,1):NEXTK
```

```
10 FORK=0TO15STEP4
15 FORL=0TO4
20 FORJ=0TO3
30 PRINTC$(K+J);D$;
35 NEXTJ:NEXTL
40 NEXTK
```

Per ottenere il codice del colore usiamo la stringa già usata nei programmi precedenti. Per fare risaltare i colori usiamo il colore bianco per lo sfondo.

La mappa colore inizia in 55296 ed occupa 1000 byte. Segue il programma COLORI4, che stampa sul video 16 righe ognuna di un colore diverso andando a scrivere con POKE il codice di un carattere nella mapppa video e il codice del colore nella corrispondente posizione della mappa colori. Questa operazione viene fatta 16 volte, cambiando ogni volta il colore dello sfondo. Ogni volta che il quadro video è completo c'è un ciclo di attesa e poi inizia la preparazione del quadro seguente.

```
1 REM COLORI4
3 B$="●":PRINT"[illegible]"
4 D=ASC(B$)
5 FORK=0TO639:POKE1024+K,D:NEXTK
6 FORM=240TO255
7 POKE53281,M
9 MV=1024:MC=55296
13 FORK=0TO15
15 FORL=0TO39
30 POKEMC+L,K
37 NEXTL:MV=MV+40:MC=MC+40
40 NEXTK
50 FORI=1TO600:NEXTI
55 NEXTM
```

Alla linea 3 abbiamo posto in B$ un carattere, e stampato una stringa di caratteri di controllo per mandare il cursore nella parte bassa del video. Alla linea 4 abbiamo calcolato il codice ASCII del carattere posto in B$. Alla linea 5 abbiamo posto nella mappa video 640 (16x40) volte il carattere di B$. Tale carattere non risulta visibile dato che la mappa colore contiene i codici del colore dello sfondo. Alla linea 6 abbiamo istituito un ciclo che fa cambiare nei 16 modi possibili il colore dello sfondo. Alla linea 7 abbiamo modificato il colore dello sfondo. Alla linea 9 abbiamo inizializzato i puntatori alle mappe video e colore. Dalla linea 13 alla linea 40 abbiamo sviluppato due cicli concatenati per mettere in ognuna delle 16 linee

video già occupate dai caratteri il codice colore e far comparire i caratteri. Il codice colore viene messo nelle posizioni della mappa colore, cambiando colore ogni 40 posizioni. Alla linea 50 abbiamo posto un ciclo d'attesa prima di tornare in 7 per efffetto del NEXT della linea 55 e ricominciare con un nuovo colore dello sfondo.

3.4 INPUT CONTROLLATO

Con le parole "input controllato" intendiamo un metodo per ricevere dati di Input dalla tastiera senza possibilità di errore, entro i limiti che precisiamo.

Quando, sia con la istruzione INPUT che con la istruzione GET, usiamo una variabile stringa per ricevere dati, vengono accettati tutti i caratteri. Questo è completamente vero se prima di rispondere alla richiesta di dati apriamo le virgolette e le chiudiamo prima di premere il tasto RETURN.
All'interno delle virgolette sono accettati anche i separatori virgola e due punti. Se non apriamo le virgolette, i caratteri virgola o due punti, sono recepiti come la fine di un campo dato e l'inizio del successivo; per questa ragione è probabile che scriviamo un numero di dati superiore al numero delle variabili disponibili nella lista di ingresso e quindi vediamo comparire sul video il messaggio: EXTRA IGNORED. In questo caso il programma prosegue, ma il video diventa poco gradevole alla vista.
Quando, invece, si usano variabili numeriche, se nei dati è presente anche un solo carattere non valido, compare il messaggio: REDO FROM START e viene chiesto nuovamente il dato (o i dati). Anche in questo caso il video non risulta gradevole. Inoltre, se abbiamo previsto di fare entrare in un quadro video un numero di righe abbastanza vicino alla capienza, pensando ogni dato non molto lungo, qualora un dato sia più lungo del previsto, il quadro video si rovina.

Da quanto detto emerge che per una buona gestione del quadro video dovremmo controllare i dati di ingresso per il tipo dei caratteri e per la lunghezza dei campi. I limiti al controllo di errore dipendono dal fatto che se invece di scrivere una parola ne scriviamo un'altra la cosa non può essere controllata e lo stesso discorso vale per i numeri.

Il controllo più rigoroso per i dati si può avere ricevendoli carattere per carattere con l'istruzione GET seguita da una variabile stringa, ma il metodo risulta un pò lento. In questo caso però si riesce a controllare ogni singolo carattere e la lunghezza del dato, e non si rischia di rovinare il quadro video.

Un altro metodo può essere quello di riservare una zona video per ricevere i dati nella parte bassa usando l'istruzione INPUT seguita da una variabile stringa; lasciare il quadro video sopra e trasferirvi il dato nella giusta posizione, dopo averne controllato la validità.

Segue il programma INPUT1, nel quale si controlla un dato numerico letto con GET.

```
1 REM INPUT1
10 REM CONTROLLO DATO NUMERICO
15 REM LETTO CON GET
20 REM NOTA LA LUNGHEZZA MASSIMA DEL DATO
25 INPUT"QUANTI CARATTERI: ";L
26 IFL=0THENSTOP
27 IFL<0THENPRINT"?":GOTO25
29 PRINT"SCRIVI DATO":GOSUB500
30 IFE=0THENPRINT"ACCETTATO: "N$:GOTO25
35 PRINT"NON VALIDO":GOTO25
500 N=0:N$="":E=0:P=0:S=0:B$=""
503 PRINT"DATO: ";
505 FORK=1TOL
510 A$="":GETA$
515 IFA$=""THEN510
517 IFA$=CHR$(13)THEN553
520 IFA$>"/"ANDA$<":"THEN550
525 IFK=1AND(A$="+"ORA$="-")THEN550
530 IFP=0ANDA$="."THENP=1:GOTO550
533 IFS=0ANDA$=CHR$(20)THEN537
534 IFS=1ANDA$=CHR$(20)THENS=0:GOTO510
535 IFS=1THEN545
536 S=1:GOTO510
537 IFB$="."THENP=0
540 PRINT"■■";:GOTO555
545 E=1:GOTO553
550 B$=A$:N$=N$+A$:PRINTA$;
551 NEXTK:PRINT:RETURN
553 K=L:NEXTK:PRINT:RETURN
555 N$=LEFT$(N$,LEN(N$)-1):GOTO510
```

All'inizio il programma chiede la lunghezza massima L del dato (nella quale devi calcolare anche il carattere RETURN), poi chiama il sottoprogramma di lettura in 500, che legge al massimo L caratteri. Il sottoprogramma ritorna il dato valido se l'indicatore E di errore risulta =0. Il dato rimane della sua lunghezza anche se <L. Il programma accetta un numero con segno e con un punto decimale; esso accetta

anche la cancellazione con il tasto DEL delle cifre numeriche e del punto decimale. Come vedi abbiamo dovuto controllare che si riceva un solo punto decimale. Osserva il controllo del tasto DEL (CHR$(20)) e del RETURN (CHR$13)) che viene recepito dall'istruzione GET, e l'uso di tre indicatori di errore: E, che viene controllato dal programma principale, P che serve per controllare la presenza di un solo punto decimale, e S che serve a cancellare il carattere precedente anche se era errato, cioè consente di non accettare il primo errore e di procedere correggendo. Ci sembra interessante esporre il diagramma a blocchi del programma INPUT1.

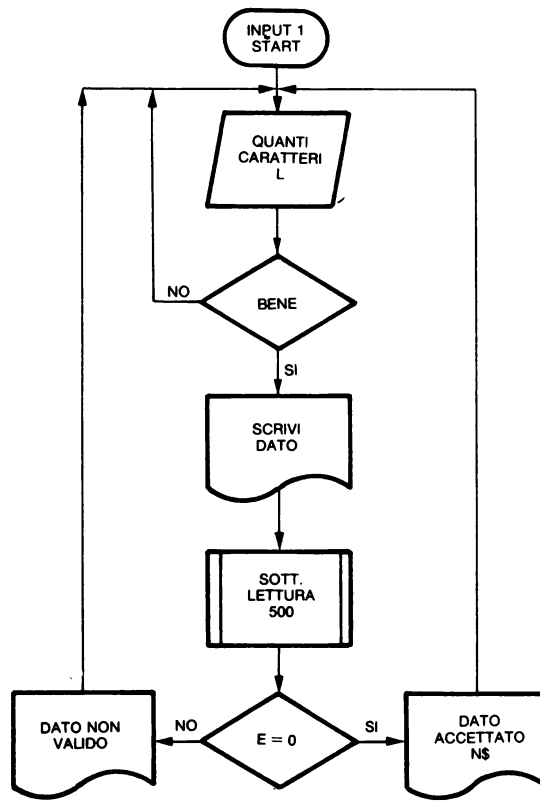
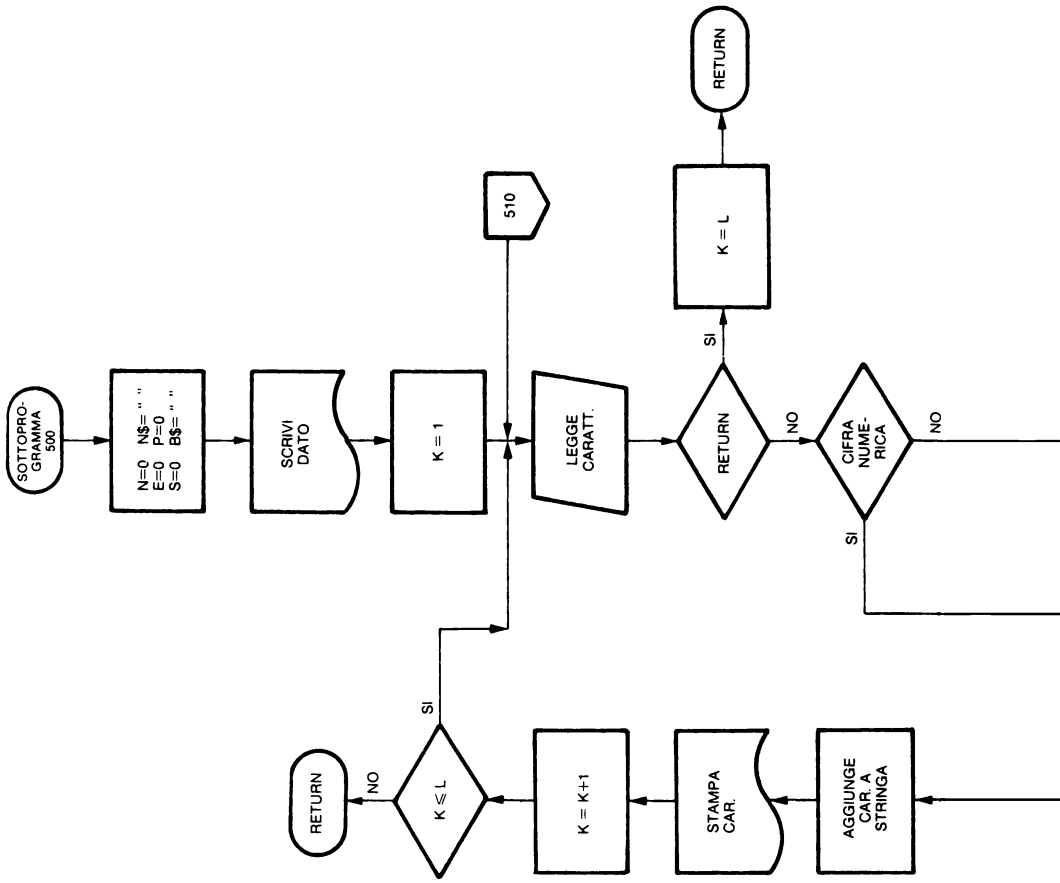


Figura 3.1 a) Diagramma a blocchi del programma INPUT1.


SOTTOPRO-
GRAMMA
500
N=0 N$= " "
E=0 P=0
S=0 B$= " "
SCRIVI
DATO
K = 1
510
LEGGE
CARATT.
RETURN
SI
NO
K = L
RETURN
CIFRA
NUME-
RICA
NO
SI
AGGIUNGE
CAR. A
STRINGA
STAMPA
CAR.
K = K+1
K ≤ L
SI
NO
RETURN

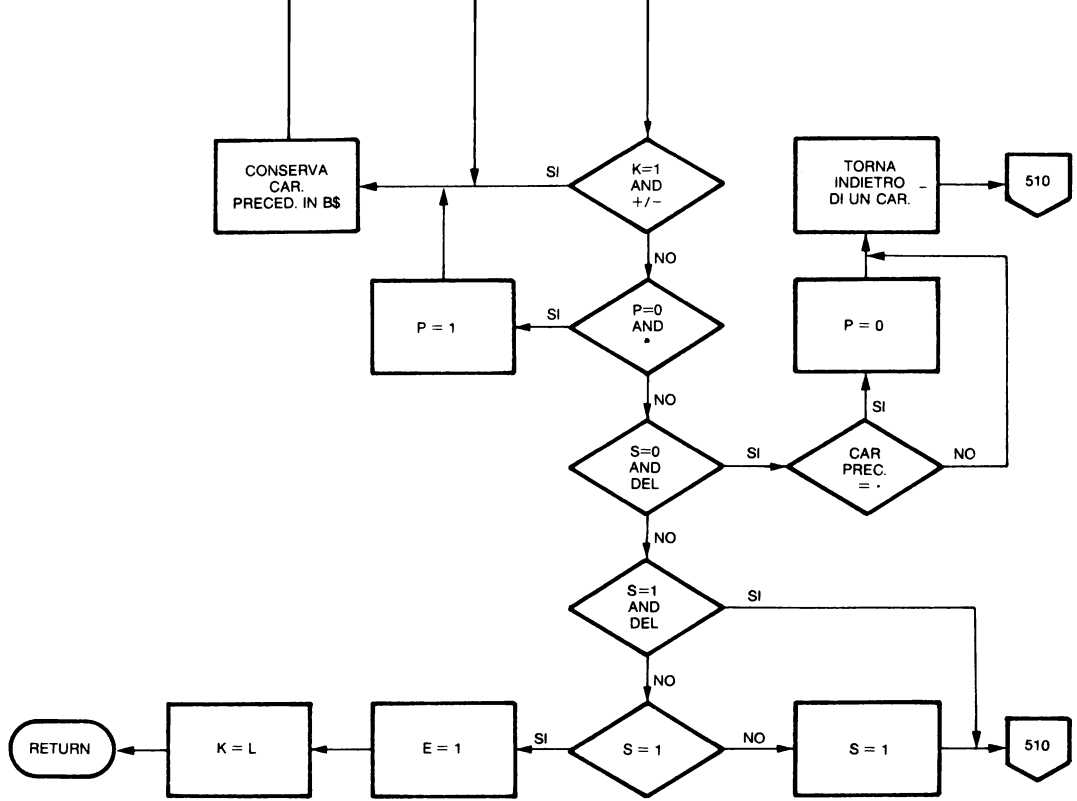


Figura 3.1 b) Sottoprogrammi INPUT1.

Segue il programma INPUT2, nel quale si controlla un dato numerico letto come stringa con INPUT.

```
1 REM INPUT2
10 REM CONTROLLO DATO NUMERICO
15 REM LETTO CON INPUT
20 REM NOTA LA LUNGHEZZA MASSIMA DEL DATO
25 INPUT"QUANTI CARATTERI: ";L
26 IFL<=0THENPRINT"?]":GOTO25
27 S$="":FORK=1TOL:S$=S$+" ":NEXTK
29 GOSUB500
30 IFE<>0THEN35
33 PRINT"ACCETTATO: "A$:GOTO40
35 PRINT"NON VALIDO"
40 PRINT"PREMI UN TASTO PER CONTINUARE"
45 Z$="":GETZ$:IFZ$=""THEN45
50 GOTO25
500 PRINT"
505 P=0:E=0:A$="":INPUT"DATO: ";A$
515 IFLEN(A$)>LTHENA$=LEFT$(A$,L)
525 FORK=1TOLEN(A$):B$=MID$(A$,K,1)
530 IFK<>1THEN540
535 IFB$="+"ORB$="-"THEN555
540 IFB$>"/"ANDB$<":"THEN555
545 IFB$="."ANDP=0THENP=1:GOTO555
550 E=E+1:K=L:NEXTK:RETURN
555 NEXTK:A$=LEFT$(A$+S$,L):RETURN
```

All'inizio il programma chiede la lunghezza massima L del dato e ritorna un dato di L caratteri con eventuale troncamento o aggiunta di spazi. Se il dato non va bene il sottoprogramma ritorna l'indicatore E di errore $< > 0$. Il dato viene letto nella parte bassa del video e riportato nella parte alta dopo l'accettazione.
Mediante l'indicatore P di errore per il punto controlliamo la presenza di un solo punto decimale. Nella fase di scrittura sotto INPUT siamo liberi di usare i tasti di controllo, tipo DEL o altri, che però sono gestiti dall'EDITOR e non fanno parte del dato ricevuto.

Segue il programma INPUT3, che controlla se un codice letto con INPUT in una variabile stringa ha uno dei 4 possibili valori.

```
1 REM INPUT3
10 DATAAC,SD,TU,XT
15 FORK=0TO3:READC$(K):NEXTK
20 INPUT"■■■CODICE: ";A$
23 S=0:FORK=0TO3
27 IFA$=C$(K)THENS=1
29 NEXTK
30 IFS=0THEN20
35 PRINT"VA BENE":STOP
```

Seguendo i suggerimenti che puoi ricavare dagli esempi riportati, potrai adattarli alle situazioni che si presenteranno nei tuoi programmi.

Naturalmente una buona routine di ingresso dati deve chiedere alla fine se i dati sono accettabili, presentandoli tutti sul video, con possibilità di andare a correggere quelli non buoni, pur essendo risultati validi da un punto di vista formale. Negli esempi del prossimo paragrafo presentiamo anche questa sequenza di programma.

3.5 PREPARAZIONE MASCHERE VIDEO

E' abbastanza raro il caso di un programma che non chieda dati di ingresso istituendo un colloquio video/tastiera. Il modo più semplice per chiedere e ricevere dati è una banalissima lista, preceduta dalla pulizia del video. Nella stesura dei programmi di questo libro ci siamo spesso limitati a questo; ora cerchiamo di fare anche qualcosa di meglio.

Cominciamo con un esempio di richiesta di data e di MENU' di scelta. La routine SCELTA che segue può essere usata in qualunque programma estendendone anche la portata.

```
1 REM SCELTA
5 CH$="■":SZ$="          "
7 CD$="■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
10 GOSUB300:PRINT"■■■"SPC(8)"DATA ODIERNA: ";
13 PRINTG$;"/";M$;"/";A$
15 GOSUB350
20 IFR$="N"THEN10
100 PRINT"■";:POKE53280,7:POKE53281,1
```

```
102 PRINTCH$;SZ$"■■■■   ■PROGRAMMA CBM 64■"
103 PRINT
105 PRINT:PRINTSZ$" ■1■ - INIZIALIZZAZIONE"
110 PRINT:PRINTSZ$" ■2■ - AGGIORNAMENTO"
115 PRINT:PRINTSZ$" ■3■ - PROCEDURA CALCOLO"
120 PRINT:PRINTSZ$" ■4■ - STAMPA RISULTATI"
125 PRINT:PRINTSZ$" ■0■ - FINE"
130 PRINT:PRINT:PRINTSZ$"      ■OPZIONE: ";
135 GETT$:IFT$<"0"ORT$>"4"GOTO135
140 PRINT"■"+T$:GOSUB360
145 A=VAL(T$):IFA=0THEN9000
150 ONAGOTO1000,2000,3000,4000
155 STOP
300 POKE53280,5:POKE53281,2
303 PRINT"■■■■■■■":OPEN1,0,1
305 PRINTSZ$"   DATA  GG,MM,AA■■■■■■■■■■";
310 INPUT#1,G$:PRINT"■■■■■■■■";
311 INPUT#1,M$:PRINT"■■■■";
312 INPUT#1,A$
313 DT$=G$+M$+A$:CLOSE1:RETURN
350 OPEN1,0,1
351 PRINTLEFT$(CD$,21);
353 PRINTSZ$"   CONFERMI ?  S■■";:INPUT#1,R$
355 IFR$<>"S"ANDR$<>"N"THEN351
357 CLOSE1:RETURN
360 A$="":PRINTCD$;
361 M$="■PREMI UN TASTO PER PROSEGUIRE■"
362 PRINT"     "M$
363 GETA$:IFA$=""THEN363
364 PRINTCH$:RETURN
1000 STOP
2000 STOP
3000 STOP
4000 STOP
9000 STOP
```

Il programma inizialmente prepara tre stringhe di costanti. La CD$ inizia con il carattere HOME e prosegue con 23 CRSR/giù; essa, se stampata tutta, porta alla linea 23 del video. Volendo ne puoi stampare una parte, usando la funzione LEFT$, e scendere alla riga desiderata.

Alla linea 10 chiamiamo il sottoprogramma 300 che richiede la data, poi la stampa e ne chiede conferma con il sottoprogramma in 350.
La routine di richiesta data usa le stringhe con il carattere CRSR/sinistra per posizionarsi su GG, MM, AA; tu devi scrivere il giorno e premere RETURN, scrivere il mese e premere RETURN e poi l'anno e premere RETURN. All'inizio essa modifica i colori del video; inoltre apre la tastiera come file per leggere (device=0) evitando il punto interrogativo di richiesta dati, così vedi solo il cursore che pulsa.
Poi il programma (da 100 a 155) propone un MENU' di scelta tra 5 opzioni diverse; osserva l'uso del campo inverso. La risposta 0 (zero) non può essere trattata con l'istruzione ON e quindi abbiamo posto un controllo alla linea 145.
I due sottoprogrammi in 350 e 360 sono di uso comune nei colloqui video/tastiera; il primo chiede conferma del quadro video e controlla la risposta S o N, il secondo crea un attesa fino alla pressione di un tasto per consentire di leggere il video.

Segue il programma QUADRO1, che è formato da due parti.
Il sottoprogramma che inizia in 107 serve per preparare una struttura di dati, con 15 campi aventi intestazioni e lunghezze diverse. E' chiaro che si tratta solo di un esempio inerente all'argomento che stiamo trattando; infatti non avrebbe senso strutturare i dati ogni volta che si usa il programma. Questo sottoprogramma dovrebbe far parte di un programma di inizializzazione che va poi a scrivere, su disco o su nastro, un file contenente le informazioni sulla struttura dei dati; tale file può poi essere riletto dai programmi e fornire i parametri necessari per gestire la struttura di dati creata.
Il sottoprogramma che inizia in 59 serve per leggere la sequenza dei dati e immagazzinarla nel vettore R$(15). Esso prevede la possibilità di correggere i dati introdotti.
Nel sottoprogramma in 107 viene proposta una struttura di dati, con intestazione e lunghezza, e viene chiesta conferma o modifica per ogni campo, con la clausola che la lunghezza di ogni campo non superi 30 caratteri e che il totale delle lunghezze dei campi non superi 237. La lunghezza di un campo non comprende il carattere RETURN. La tastiera viene aperta come file per abolire il punto interrogativo in fase di Input; ricorriamo alla stringa CD$ e alla funzione TAB per posizionarci nei punti desiderati del video. Il vettore D$(15) contiene le descrizioni dei campi e il vettore L(15) le lunghezze. Aver disposto tutto in variabili con indice consente di programmare facilmente cicli.

```
1 REM QUADRO1
3 L=15:DIMD$(15),L(15),M$(11),M(11),R$(15)
5 CH$="":CD$="
7 SZ$="          "
9 PRINT"";:FORI=1TO30:SP$=SP$+" ":NEXTI
11 DATA8,6,9,0,5,5,6,8,0,4,9
13 FORK=1TO11:READM(K):NEXTK
15 M$(1)="SCELTA NOMI DEI 15 CAMPI"
17 M$(2)="CONFERMA IL NOME SE VA BENE"
19 M$(3)="ALTRIMENTI RISCRIVILO."
21 M$(4)=""
23 M$(5)="OSSERVA LE LUNGHEZZE DEI CAMPI"
25 M$(6)="PUOI CAMBIARLE. LA SOMMA DELLE"
27 M$(7)="LUNGHEZZE DEI CAMPI NON PUO'"
29 M$(8)="SUPERARE 237 CARATTERI."
31 M$(9)=""
33 M$(10)="LA LUNGHEZZA DI UN CAMPO NON PUO'"
35 M$(11)="SUPERARE 30 CARATTERI."
37 MS$="       PREMI UN TASTO PER PROSEGUIRE"
39 MT$="         QUALE CAMPO      "
41 PRINTCH$;:GOSUB107
43 GOSUB59:STOP
45 A$="":PRINTCD$;MS$
47 GETA$:IFA$=""THEN47
49 PRINTCH$:RETURN
51 CLOSE1:OPEN1,0,1:PRINTLEFT$(CD$,21);
53 PRINTSZ$"   CONFERMI    S";
55 INPUT#1,R$:IFR$<>"S"ANDR$<>"N"THEN51
57 CLOSE1:RETURN
59 CLOSE1:OPEN1,0,1
61 PRINT"  INGRESSO DATI"
63 F=0:PRINT" 1";TAB(4);D$(1)+":←";
65 INPUT#1,R$(1):PRINT
67 IFLEFT$(R$(1),1)="←"THENF=1:RETURN
69 FORI=2TOL:PRINTI;TAB(4);D$(I)":←";
71 INPUT#1,R$(I):PRINT:NEXTI:GOSUB85
73 GOSUB51:IFR$="S"THENCLOSE1:RETURN
75 CLOSE1:OPEN1,0,1
77 PRINTLEFT$(CD$,21);MT$;:INPUTR
79 IFR<1ORR>LTHEN77
81 IFR>LORR<1GOTO77
83 GOSUB91:GOTO73
```

```
85 FORI=1TOL
87 R$(I)=LEFT$(R$(I)+SP$,L(I))
89 NEXTI:RETURN
91 PRINT"[illegible]";:FORI=1TOR-1
93 PRINTI;TAB(4);D$(I);":";R$(I):NEXTI
95 PRINTR;TAB(4);D$(R);":←[illegible]";:INPUT#1,R$(R)
97 PRINT:R$(R)=LEFT$(R$(R)+SP$,L(R))
99 IFR>=LTHEN105
101 FORI=R+1TOL
103 PRINTI;TAB(4);D$(I);":";R$(I):NEXTI
105 RETURN
107 DW$="S":D$(1)="COGNOME":D$(2)="NOME"
109 D$(3)="INDIRIZZO":D$(4)="TELEFONO"
111 D$(5)="C.A.P.":D$(6)="CITTA'"
113 FORIK=7TO15:D$(IK)="C"+CHR$(42+IK):NEXTIK
115 DATA20,15,30,15,8,15,22,22,15,15
117 DATA15,15,15,10,5
119 FORIK=1TO15:READL(IK):NEXTIK
121 CLOSE1:OPEN1,0,1:PRINTCH$;
123 FORK=1TO11
125 PRINTSPC(M(K))"[illegible]"M$(K)"[illegible]":NEXTK
127 GOSUB45
129 PRINTCH$"[illegible]CONFERMA STRUTTURA DATI[illegible]"
131 FORJ=1TO15
133 PRINT"L=";L(J);TAB(10);D$(J):NEXTJ
135 PRINT:PRINT"SE VA BENE PREMI RETURN"
137 PRINT"SE NO RISCRIVI PER SOSTITUIRE"
139 FORJ=1TO15
141 PRINTLEFT$(CD$,J+1);TAB(10);"[illegible]";D$(J);"[illegible]";
143 PRINTCD$;SP$;CD$;:T$="":INPUT#1,T$
145 IFT$=""THEN149
147 D$(J)=T$
149 PRINTLEFT$(CD$,J+1);TAB(8);SP$;
151 PRINTLEFT$(CD$,J+1);TAB(10);D$(J)
153 PRINTLEFT$(CD$,J+1);TAB(2);"[illegible]";L(J);"[illegible]";
155 PRINTCD$;SP$;CD$;:IL=0:INPUT#1,IL
157 IFIL=0THEN165
159 IFIL>30THEN153
161 L(J)=IL
163 PRINTLEFT$(CD$,J+1);TAB(4);"  ";
165 PRINTLEFT$(CD$,J+1);TAB(2);L(J):NEXTJ
167 YX=0:FORJ=1TO15:YX=YX+L(J):NEXTJ
```

```
169 IFYX<=237THEN177
171 PRINTLEFT$(CD$,20);
173 PRINT"TROPPI CARATTERI";YX
175 GOSUB45:GOTO129
177 GOSUB51
179 IFR$="N"THEN121
181 CLOSE1:RETURN
```

Nel sottoprogramma in 59 si apre ancora la tastiera come file e si leggono i dati con INPUT # tutti come stringhe. Non abbiamo svolto un controllo dei campi, ma potrebbe essere introdotto per particolari valori degli indici, per controllare, come abbiamo già visto nel precedente paragrafo, i campi numerici o altro. Il sottoprogramma consente la modifica dei dati introdotti. Per uscire dal sottoprogramma, nel caso faccia parte di un programma che legge dati ciclicamente, abbiamo posto il controllo sul primo campo; se si risponde con RETURN resta valido il carattere "freccia sinistra" e viene riconosciuto.
Il sottoprogramma in 45 serve per creare un ciclo di attesa fino alla pressione di un tasto. Quello in 51 chiede conferma del quadro video presente.

Segue il programma QUADRO2, nel quale abbiamo cercato di realizzare un quadro video più bello, giocando sui colori e sui riquadri in campo inverso.

```
1 REM QUADRO2
4 V$="▒▒▒
```

```
124 K=VAL(R$):IFK<10ORK>12THEN115
127 POKE646,2
130 PRINTLEFT$(V$,CR(K)+2)TAB(CC(K));"▒ ▒▒▒▒"
133 GOSUB154
136 GOTO97
139 POKE646,14:PRINT"▒        DATI LETTI"
142 FORK=1TO12
145 PRINTD$(K);": ";R$(K)
148 NEXTK
151 STOP
154 POKE198,0:I=1:R$(K)=""
157 A$="":GETA$:IFA$=""THEN157
160 IFA$=CHR$(13)THEN175
163 PRINTA$;
166 R$(K)=R$(K)+A$
169 IFI=L(K)THEN175
172 I=I+1:GOTO157
175 POKE198,0:FORI=1TO200:NEXTI:RETURN
```

Anche qui ricorriamo ai vettori per memorizzare intestazioni (D$(12)), lunghezze (L(12)) e valori (R$(12)) dei dati. In più abbiamo creato un vettore CR(12), per contenere il numero della riga video dove vogliamo posizionare il dato, e un vettore CC(12) per contenere il numero da usare per la posizione sulla riga. Questi due vettori servono, per la funzione LEFT$ che preleva la parte necessaria della stringa V$ e manda per mezzo dell'istruzione PRINT alla riga desiderata, e per la funzione TAB che crea la posizione sulla riga. Il quadro video viene preparato sfruttando il cambio dei colori e il campo inverso. Dopo aver posto sul video le descrizioni e la numerazione di ogni campo, creiamo nella riga sottostante un rettangolo in campo inverso per contenere il dato, esattamente della lunghezza del dato da ricevere. Quest'ultimo accorgimento aiuta a non scrivere dati troppo lunghi. I dati sono letti con GET carattere per caratere dal sottoprogramma in 154. Non è stato necessario aprire la tastiera come file, dato che GET non crea il punto interrogativo.
Anche qui i dati possono essere corretti scegliendo il campo da correggere.

La lunghezza dei campi può essere quella stabilita o può essere minore premendo il RETURN. Il programma non accetta campi più lunghi dei limiti contenuti in L(K). Osserva l'uso di POKE 198,0 per svuotare il buffer della tastiera, e dei cicli d'attesa con FOR per rallentare le operazioni.

Il sottoprogramma in 154 non fa controlli sui caratteri introdotti, salvo per la lunghezza dei campi e il RETURN. Devi stare attento all'effetto che può provocare l'uso del tasto DEL, che viene accettato come un qualunque carattere, produce sul video l'effetto di cancellazione, ma influisce sul computo del numero dei caratteri. La routine in 154 può essere modificata, nel modo mostrato nel precedente paragrafo, o in altro, per fare accettare solo determinati gruppi di caratteri.

INDICE

Capitolo 1 - MEMORIZZAZIONE E CARICAMENTO PROGRAMMI
1.1 Introduzione 1
1.2 File di programmi su cassetta 2
1.3 Comandi per la gestione del disco 4
1.4 File di programmi su disco 11
1.5 Overlay di programmi 12
1.6 Programmi e suggerimenti utili 16

Capitolo 2 - I FILE SU STAMPANTE
2.1 Introduzione 19
2.2 Istruzioni di stampa 22
2.3 Modi di stampa e uso dei caratteri di controllo 27
2.4 Gestione del buffer di stampa 47
2.5 Come si compone uno stampato 49
2.6 Copia del video su carta 53

Capitolo 3 - COSTRUZIONE DI PROGRAMMI
3.1 Introduzione 59
3.2 Algoritmi di ordinamento 60
3.3 Caratteri a caso e copia su carta 72
3.4 Stampa di una fattura professionale 77

Capitolo 4 - CODICI E NUMERI DEL CALCOLATORE
4.1 Premessa 87
4.2 Numeri interi 87
4.3 Numeri reali 90
4.4 Codici ASCII e D/CODE - Caratteri stampabili 95
4.5 Codici di controllo 118
4.6 Codifica parole chiave - Tokens 120

CAPITOLO 1

MEMORIZZAZIONE E CARICAMENTO PROGRAMMI

1.1 INTRODUZIONE

In questo Capitolo ci occupiamo della gestione dei programmi relativamente ai supporti di memorizzazione cassetta e disco e all'utilizzo della memoria.

Il linguaggio Basic dispone dei comandi LOAD, SAVE e VERIFY, che consentono di caricare un programma in memoria, di memorizzarlo sul supporto e di verificare se la registrazione è buona.
Per eseguire questi comandi non è necessario stabilire una comunicazione con la periferica per mezzo del comando OPEN e quindi non serve neanche il comando CLOSE.
Per quanto riguarda il disco vedremo anche alcuni comandi particolari che consentono operazioni di servizio sui file di programmi.
Rimandiamo al volume che seguirà subito questo la trattazione di tutta la problematica relativa al trattamento dei file di dati su disco e cassetta.
Abbiamo pensato di farti cosa gradita aggiungendo i Paragrafi 1.5 e 1.6, ma te ne raccomandiamo la lettura solo se sei già diventato un po' esperto di programmazione.

Ti consigliamo, se desideri sfruttare in pieno le possibilità del tuo calcolatore, di acquistare uno di quei programmi di utilità (tipo TOOL-KIT, o simili) che consentono di:
...numerare automaticamente le linee del programma mentre si scrive;
...rinumerare un programma che abbia subito molti aggiustamenti;
...cancellare blocchi di programma;
...ricercare una particolare stringa di caratteri nel programma;
...sostituire automaticamente una stringa di caratteri con un'altra;
...assegnare funzioni utili ai tasti funzione posti sulla destra della tastiera;
...ricercare errori;
...visualizzare i numeri delle linee di programma che si eseguono;
...fondere programmi diversi in un unico programma.

Abbiamo trattato la struttura dei programmi nella memoria del calcolatore nell'ottavo Volume: qui ci basta ricordare che ogni linea di programma contiene due byte, chiamati LINK, che puntano all'indirizzo della prossima istruzione. I programmi Basic, in condizioni normali, sono situati in memoria a partire dal byte di indirizzo 2049. I valori di LINK nascono in conseguenza. Noi possiamo spostare il puntatore all'inizio del programma Basic, agendo sui byte 43 e 44 della pagina 0 della RAM. Se carichiamo un programma, dopo aver operato questo spostamento, esso viene RILOCATO, cioè viene caricato a partire dal nuovo indirizzo e i due byte di LINK vengono modificati.
Nei comandi SAVE e LOAD si può aggiungere un parametro, come SA, dopo il DN della periferica, per intervenire sulla rilocazione. L'effetto è il seguente:
..SAVE con SA=1 memorizza il programma in modo tale che in fase di caricamento con un LOAD senza SA il programma non viene rilocato;
..LOAD con SA=1 carica il programma salvato senza SA senza rilocarlo.
Devi usare con cautela queste opzioni; infatti dopo aver caricato un programma che non inizia nel modo solito, dovrai preoccuparti di modificare il puntatore (byte 43 e 44) per poterlo far partire.

Nel seguito i parametri compresi nella spiegazione delle istruzioni possono comparire sia in lettere minuscole che maiuscole.

1.2 FILE DI PROGRAMMI SU CASSETTA

Per memorizzare un programma presente in memoria su cassetta possiamo scrivere:

1) SAVE

2) SAVE ""

3) SAVE "nome"

4) SAVE "nome",1

5) SAVE "nome",1,1

6) SAVE "nome",1,2

7) SAVE "nome",1,3

che sono solo alcuni casi particolari del formato:

SAVE [fn] [,dn] [,sa]

dove:
..fn è il nome del file e può mancare, se presente vengono accettati solo i primi 16

caratteri;
..dn è il numero della periferica, che per la cassetta è 1, e può essere omesso;
..sa è un parametro che influisce sulla rilocazione dei programmi se è posto a 1, come già abbiamo visto nel precedente paragrafo; per la cassetta esso può assumere anche il valore 2, che significa scrittura con segnalazione finale di fine nastro (EOT), e il valore 3 che significa sia non rilocabile che con EOT finale.

Si possono memorizzare i programmi senza nome, ma a nostro avviso, non è consigliabile. E' sempre meglio assegnare un nome al programma e riportarlo sulla etichetta esterna della cassetta.

Ti raccomandiamo di mantenere il registratore sempre in stato di riposo, senza tasti abbassati, questo per evitare che parta senza aver prima sistemato la cassetta. Se vuoi controllare dove arriva la registrazione, devi azzerare il contagiri. Inoltre è meglio riavvolgere la parte iniziale del nastro non utilizzabile prima di inserire la cassetta nel registratore.

Quando premi il tasto RETURN per fare accettare il comando, o quando esso va in esecuzione da programma, compare sul video il messaggio:

PRESS RECORD & PLAY ON TAPE

a quel punto devi premere PRIMA il tasto RECORD e POI il tasto PLAY, assolutamente no il viceversa. Infatti la registrazione inizia quando si preme PLAY, e, se il tasto RECORD non è premuto, si rischia di perderne una parte.
Durante la registrazione scompaiono le scritte dal video e riappaiono quando essa è terminata. Il sistema registra due volte i blocchi di programma; in fase di lettura controlla che non ci siano errori e differenze tra i due blocchi.

Dopo aver memorizzato un programma su nastro è bene farne la verifica. Puoi scrivere:

VERIFY "nome",1 o anche solo VERIFY

naturalmente devi riavvolgere il nastro al punto giusto. Per evitare confusioni è bene usare il nome che si è assegnato al programma; infatti, se non hai riavvolto bene il nastro e sei andato troppo indietro, viene ricercato il programma, da confrontare con quello in memoria, mediante il nome.
Dopo il comando compare la richiesta:

PRESS PLAY ON TAPE

anche in questo caso quando hai accontentato la richiesta, scompaiono le scritte dal video, per ricomparire alla fine con il messaggio di OK o di errore.

Il programma viene registrato su nastro esattamente come si trova in memoria, byte dopo byte.

Per caricare in memoria un programma da cassetta si può scrivere:

1) LOAD ""

2) LOAD "nome"

3) LOAD "nome",1

4) LOAD "nome",1,1

che sono solo alcuni casi particolari del formato:

LOAD [fn] [,1] [,1]

dove i parametri hanno il significato già visto.

Se manca il nome viene caricato il primo programma che viene incontrato sul nastro.

Dopo l'accettazione del comando compare la richiesta:

PRESS PLAY ON TAPE

e quando esegui scompaiono le scritte dal video; dopo un pò ricompaiono le scritte e c'è la segnalazione:

FOUND...

se qualcosa è stato trovato. Dopo scompaiono nuovamente le scritte e viene caricato il programma. Al termine ricompaiono le scritte con segnalazione LOADING/READY se è andato tutto bene, oppure di errore.

1.3 COMANDI PER LA GESTIONE DEL DISCO

L'unità 1541 o la 1571 collegata al COMMODORE 64 consente di usare i floppy disk come supporti di memorizzazione.
I floppy disk sono un buon supporto di memorizzazione ed arricchiscono le possibilità del calcolatore.
Per ora qui ci limitiamo a fornirti le informazioni necessarie per poter memorizzare programmi su disco e per poterli ricaricare in memoria.

I dischetti devono essere maneggiati con cura, senza piegarli e senza toccare le parti visibili dalle aperture del contenitore. Quando si acquista un dischetto nuovo, esso non può esssere adoperato se non si esegue su di esso l'operazione di FORMATTAZIONE, che vedremo tra poco.

Sull'unità, a sinistra, è presente un indicatore luminoso; esso diventa verde quando si dà corrente. Quando è in esecuzione un'operazione disco, l'indicatore diventa rosso, e, al termine torna verde. Se vedi pulsare in rosso l'indicatore significa che l'operazione in corso ha difficoltà ad essere eseguita. In certi casi compare anche un messaggio di errore dopo molti tentativi di esecuzione. Conviene spegnere l'unità, aspettare un breve intervallo di tempo e poi riprovare; se le condizioni di errore persistono può esserci un guasto nell'unità, ma può anche essere rovinato il dischetto.

All'interno dell'unità 1541 è presente un microprocessore con 16K di memoria ROM e 2K di memoria RAM. La memoria ROM contiene il Sistema Operativo del disco (DOS - Disk Operating System) che provvede all'esecuzione delle operazioni che vengono lanciate dal calcolatore verso la periferica. La memoria RAM serve per i buffer necessari a contenere le informazioni che vanno scritte sul floppy o che sono lette da esso, e l'area di lavoro del DOS.

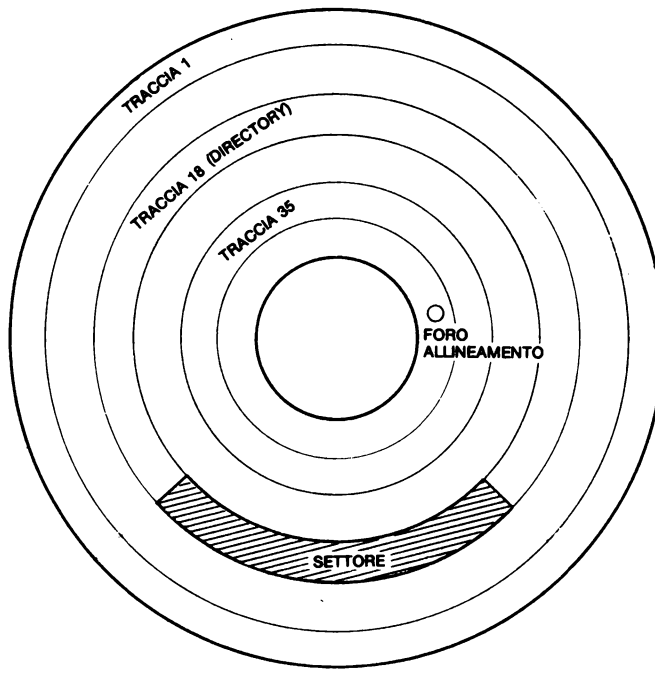


Figura 1.1 Schema di un floppy disk.

Nella figura 1.1 è riportato uno schema semplificato del dischetto. Puoi vedere una parte tratteggiata con la dicitura SETTORE; questa è la parte di dischetto che viene coinvolta in una operazione di lettura o scrittura. In un SETTORE o BLOCCO sono contenuti 256 caratteri (256 byte) disponibili per l'utente.
Nella figura sono segnate le tracce; 1, 35 e 18; di queste tracce concentriche sul floppy ne abbiamo disponibili 35. Nella figura è segnata la prima e l'ultima; la numero 18 è accompagnata dal nome "directory", infatti essa è una traccia un pò speciale e serve per registrare l'indice del contenuto del disco (un pò come l'indice di un libro).
Il numero di settori disponibili su ogni traccia non è sempre uguale, varia per gruppi di tracce. La ripartizione è la seguente:

Num. traccia	Quanti settori	Num. settori
da 1 a 17	21	da 0 a 20
da 18 a 24	19	da 0 a 18
da 25 a 30	18	da 0 a 17
da 31 a 35	17	da 0 a 16

Il numero totale di settori disponibili risulta:

21x17+7x19+6x18+5x17=683

ma di questi, i 19 della traccia 18 sono riservati al sistema, e quindi restano disponibili:

683-19=664 settori.

Ogni settore contiene 256 byte; in totale un floppy disk mette a disposizione:

664x256=169984 byte.

Per poter usare il dischetto su di esso devono essere registrate alcune indicazioni; esse sono gli indirizzi delle tracce e dei settori e le informazioni iniziali della DIRECTORY. L'operazione che fa queste registrazioni iniziali si chiama FORMATTAZIONE. Si può formattare un disco nuovo e si ottiene un disco pronto per l'uso. Si può anche formattare un disco già usato, si ottiene un disco pronto per l'uso che ha perso tutte le informazioni che conteneva precedentemente.

Il FORO DI ALLINEAMENTO che compare nella figura serve per allineare il dischetto rispetto alla testina di lettura e scritura dell'unità, al momento dell'introduzione. Questo allineamento avviene automaticamente quando si introduce il dischetto nell'unità, ma può anche essere ottenuto con un comando da programma. Il comando prende il nome di INIZIALIZZAZIONE.

Il DOS esegue le operazioni di lettura e scrittura relative al disco e gestisce l'indice del disco. In sostanza esso gestisce la comunicazione tra il disco e l'interfaccia seriale che lo collega al calcolatore.

Per stabilire una comunicazione con il disco il calcolatore può fare le seguenti cose:

.. inviare un comando di LOAD;
.. inviare un comando di SAVE;
.. inviare un comando di OPEN.

Delle operazioni di LOAD e SAVE ci occupiamo nel prossimo paragrafo; vediamo qui cosa succede con OPEN.

Il formato della OPEN è:

OPEN lfn,dn,sa

dove:

lfn, deve essere il numero logico del file che si vuole usare per comunicare;

dn, è il numero della periferica, e, nel caso di una sola unità collegata, risulta 8;

sa, indirizzo secondario, prende in questo caso il nome di CANALE; esso può variare da 2 a 15.

In realtà per la periferica sono disponibili i numeri di canale da 0 a 15, ma 0 e 1 sono SA usati dalle istruzioni LOAD e SAVE, quindi la OPEN può usare i numeri da 2 a 15. Precisamente i numeri da 2 a 14 per le operazioni relative ai file di dati, che non trattiamo qui, il 15 per la trasmissione dei comandi.

Prendiamo in considerazione la:

OPEN N,8,15 usata per impartire comandi al disco. N può essere scelto a piacere, noi, per abitudine, usiamo il numero 15.

Naturalmente dopo aver impartito comandi come vedremo a seguito di questa OPEN, è necessario usare il comando CLOSE per chiudere la comunicazione; esso si scrive:

CLOSE N

usando lo stesso lfn usato nella OPEN.

A seguito della OPEN impartiamo i comandi con una stringa nella istruzione:

PRINT# lfn,stringa-comandi

I comandi che possono essere impartiti sono i seguenti:

.. NEW	formattazione dischetto;
.. INITIALIZE	inizializzazione dischetto;
.. VALIDATE	controllo e sistemazione dischetto;
.. COPY	copia di un file;
.. RENAME	cambio nome di un file;
.. SCRATCH	cancellazione di un file.

Per indicare un comando è sufficiente indicare la prima lettera.

Esaminiamo ora dettagliatamente ogni comando. Sono possibili due procedure:
.. usare la stringa comando con una istruzione PRINT# ;
.. usare la stringa comando direttamente nella OPEN, facendola seguire dopo gli altri parametri.

Comando NEW

Prepara un disco nuovo per l'uso o cancella un disco già usato e lo rende nuovamente disponibile. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"N0:FN,XX"

OPEN15,8,15,"N0:FN,XX"

FN è il nome da assegnare al disco e può essere lungo al massimo 16 caratteri. XX è un codice a due caratteri di identificazione del disco; esso non può essere omesso se si sta formattando un disco nuovo. La presenza dei due caratteri di identificazione fa sì che vengano scritti sul disco gli indirizzi di traccia e settore. Se si formatta un disco già usato e si omette l'identificazione, viene mantenuta la precedente identificazione, può essere cambiato il nome e il disco viene cancellato impiegando un tempo minore; infatti non vengono riscritti tutti gli indirizzi di traccia e settore. Lo zero dopo N identifica il disco; se si usasse un'altra periferica con due dischi, si avrebbe disco 0 e disco 1.

Comando INITIALIZE

Serve per caricare nella RAM dell'unità le informazioni necessarie per gestire il

disco e ad allineare fisicamente il dischetto rispetto alla testina di lettura e scrittura. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"I"

OPEN15,8,15,"I"

dove a I si può volendo far seguire 0, come numero del disco, ma non è necessario avendo una unità disco semplice (non con due dischetti).

Per una buona gestione del disco, esso deve essere inizializzato. Il sistema lo fa automaticamente al momento della introduzione del dischetto, se osservi l'unità vedi accendersi la luce rossa per un brevissimo tempo e senti girare il dischetto. Ti consigliamo di inserire nei tuoi programmi, soprattutto se prevedono l'operazione di cambio dei dischetti, il comando di inizializzazione.

E' importante non confondere i due comandi di formattazione e inizializzazione; il primo scrive materialmente qualcosa sul dischetto, il secondo scrive qualcosa nella RAM e fa girare il dischetto per sistemarlo nell'alloggiamento.
I dischetti si pongono in rotazione solo quando svolgono operazioni di lettura o scrittura, altrimenti sono in stato di quiete.

Comando VALIDATE

Serve per mettere ordine in un disco dove siano presenti delle situazioni irregolari, tipo file non chiusi. Il comando sistema le zone libere o occupate sul dischetto leggendo il contenuto dell'indice (Directory). Non si può usare se il disco ha registrazioni non riportate nella Directory. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"V"

OPEN15,8,15,"V"

dove si può volendo far seguire a V il numero 0 per il disco.

Un disco non è sicuramente in buone condizioni se la somma dei settori liberi e occupati rilevabile dalla lista della Directory non dà 664.

Comando COPY

Consente di copiare un file assegnandogli un altro nome (un programma memorizzato due volte con nomi diversi). Sequenze possibili:

OPEN15,8,15:PRINT# 15,"C:DFN=SFN"

OPEN15,8,15,"C:DFN=SFN"

dove:

DFN, è il nome del file destinazione (la copia che si vuole avere);
SFN, è il nome del file sorgente (quello che si vuole ricopiare).
I nomi possono essere al massimo di 16 caratteri.

Comando RENAME

Consente di cambiare il nome ad un file; viene cambiato il nome nella Directory senza spostare il file da dove si trova sul dischetto. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"R:NFN=OFN"

OPEN15,8,15,"R:NFN=OFN"

dove:
NFN, è il nuovo nome che si vuole asssegnare al file;
OFN, è il vecchio nome del file.

Anche qui si può usare dopo R il numero 0 per il disco.

Comando SCRATCH

Serve per cancellare file dal dischetto. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"S:FN"

OPEN15,8,15,"S:FN"

Per leggere nella memoria del calcolatore la Directory del disco si usa il comando:

LOAD"$",8

e per vederne il contenuto si scrive:

LIST.

Per listare la Directory sulla stampante si scrive:

OPEN4,4:CMD4:LIST

e poi, dopo la lista:

PRINT# 4:CLOSE4.

Naturalmente dopo avere usato i comandi descritti resta aperta la comunicazione tramite il canale 15; per chiuderla devi eseguire: CLOSE15.

Quando si usano comandi che richiedono nomi di file FN, si possono usare le seguenti semplificazioni:

* per rimpiazzare la parte finale di un nome, non importa di quanti caratteri;

? per rimpiazzare un carattere anche all'interno di un nome.

PRINT#15,"S0:PIP*" cancella tutti i file che hanno il nome che inizia con PIP.

PRINT#15,"S0:AE??GH" cancella tutti i file che hanno un nome di 6 caratteri, che inizia con AE e termina con GH.

1.4 FILE DI PROGRAMMI SU DISCO

Per la gestione dei programmi sul disco sono disponibili gli stessi comandi che per la cassetta, solo che in questo caso è obbligatorio citare il numero della periferica, DN. Un'altra differenza è che non viene chiesto di avviare l'unità; se non è stato inserito il dischetto si vede pulsare la luce di errore.
I comandi vanno dati dopo che il dischetto è stato inizializzato con il comando I, soprattutto se esso è stato appena cambiato.

Per memorizzare un programma su disco si scrive:

SAVE"0:FN",8

dove:
0, può essere omesso (numero disco);
FN, è il nome da assegnare al file programma;
8, è il numero della periferica.

Se fai seguire a 8 la virgola e il numero 1 (...,8,1), ottieni di salvare il programma in

modo che al momento del LOAD non venga rilocato.

Si può scrivere il comando come segue:

SAVE"@0:FN",8

per ottenere che il file, eventualmente già presente con lo stesso nome, venga cancellato e sostituito da quello nuovo.
Noi preferiamo seguire la procedura di cancellare prima con il comando SCRATCH il vecchio file, e poi usare il comando SAVE senza il carattere @.

Dopo il SAVE è raccomandabile l'uso del comando di verifica, che si scrive:

VERIFY"0:FN",8 oppure

VERIFY"*",8

e produce l'effetto di confrontare il programma di nome FN, oppure l'ultimo memorizzato, con quello presente in memoria.Se la verifica non dà buon esito si deve cancellare la registrazione e ripetere la procedura di SAVE dopo aver nuovamente inizializzato il dischetto.

Per caricare un programma da disco si scrive:

LOAD"0:FN",8

Come sempre il numero 0 per il disco può essere omesso.
Se il programma non esiste sul dischetto si vede pulsare la luce rossa; in questo caso devi inizializzare nuovamente il dischetto, leggere la Directory per controllare i nomi dei programmi presenti ed eventualmente cambiare dischetto.
Se il caricamento è terminato senza segnalazione di errore puoi far partire il programa o con RUN o con GOTON.
L'operazione di LOAD chiude tutti i canali di comunicazione tra calcolatore e disco; dovrà provvedere il programma a aprire i canali necessari con OPEN.
Usando dopo 8 lo SA uguale a 1 ottieni di non rilocare il programma che carichi.

1.5 OVERLAY DI PROGRAMMI

Quando si carica un programma usando il comando LOAD da tastiera si opera praticamente un NEW del calcolatore, cioè si azzera tutto prima di caricare il programma. Inoltre il programma parte a comando da tastiera con RUN o con GOTO.

Quando, invece, il comando LOAD viene eseguito da un programma non si ha l'azzeramento e il programma parte automaticamente, per effetto del nuovo caricamento il vecchio programma viene cancellato.
Quando si programma una procedura complessa che richiede parecchi programmi, è comodo poter alternare in memoria i programmi senza perdere le variabili, ma facendo in modo che un programma ritrovi le variabili lasciate dall'altro. Per ottenere questo bisogna far iniziare le variabili sempre dallo stesso indirizzo di memoria indipendentemnte dalla lunghezza dei programmi.
L'alternanza dei programmi in memoria prende il nome di OVERLAY.
Per risolvere il problema nel migliore dei modi si deve considerare il programma più lungo e rilevare il valore dei puntatori all'inizio delle variabili, aumentare di poco per lasciare un pò di margine per eventuali modifiche del programma e prendere nota dei valori trovati.
Si deve poi creare un programma di lancio della procedura; in esso con delle istruzioni POKE si modificano i tre puntatori di inizio variabili, inizio array e fine array, che partono inizialmente dagli stessi valori. L'ideale sarebbe anche inizializzare nel programma di lancio tutte le variabili che usa la procedura, in modo da fissarle in memoria, salvo i corpi delle stringhe, naturalmente; questo fa guadagnare velocità ai programmi. Il programma di lancio esegue il LOAD del primo programma della procedura, questo poi esegue il LOAD del secondo e così via; le variabili non vengono toccate e tutti i programmi le possono usare.
Si devono usare degli accorgimenti, per esempio non possono mettersi in comune stringhe definite all'interno delle linee di un programma; infatti per stringhe di questo tipo la testata si trova tra le altre variabili, ma il puntatore manda all'interno del programma e quando se ne carica un altro, a quell'indirizzo si trovano altre cose. Le variabili con indice devono essere dimensionate nel programma di lancio.

Riportiamo un semplice esempio; tre programmi: LANCIO, CORTOMOD e LUNGOMOD. Il programma LANCIO sposta i tre puntatori molto avanti, anche se i tre programmi esempio sono corti, e carica CORTOMOD. Questo chiede una stringa di Input, poi stampa le sue variabili alla stampante e carica LUNGOMOD. Questo stampa le variabili ereditate dal programma precedente e poi le sue.

```
1 REM LANCIO CORTOMOD
5 POKE45,0:POKE46,16
6 POKE47,0:POKE48,16
7 POKE49,0:POKE50,16
8 PRINT"CARICO PROGRAMMI"
10 LOAD"CORTOMOD",8
15 END
```

```
1 REM CORTOMOD
2 C$="SCRIVI CC$:"
3 PRINTC$;:INPUTCC$
5 OPEN4,4:CMD4
6 PRINT"RUN CORTOMOD"
10 A=123
15 B=678
25 PRINT"A=";A,"B=";B
26 PRINT"CC$=";CC$
27 PRINT#4:CLOSE4
30 LOAD"LUNGOMOD",8
```

```
1 REM LUNGOMOD
2 REM PROVA CONSERVAZIONE VARIABILI
3 REM A,B,C$ DEVONO MANTENERSI
5 OPEN4,4:CMD4
6 PRINT"RUN LUNGOMOD"
10 D=998
15 E=790
20 F$="LUNGOMOD"
25 PRINT"A=";A,"B=";B
26 PRINT"CC$=";CC$
28 PRINT"D=";D,"E=";E
29 PRINT"F$=";F$
30 PRINT#4:CLOSE4
40 END
```

RISULTATI PROGRAMMI PRECEDENTI

```
RUN CORTOMOD
A= 123              B= 678
CC$=CORTOMOD
```

```
RUN LUNGOMOD
A= 123              B= 678
CC$=CORTOMOD
D= 998              E= 790
F$=LUNGOMOD
```

Dopo aver provato questi programmi devi ripristinare i valori dei puntatori con delle istruzioni POKE o spegnere il calcolatore per cancellare la memoria.

Per fare una controprova riportiamo i due programmi CORTO e LUNGO, che non modificanno i puntatori, dai risultati puoi vedere che il secondo programma, più lungo del primo, invade la zona variabili del primo e quindi non esistono più le prime variabili.

```
1 REM CORTO
5 OPEN4,4:CMD4:PRINT"RUN CORTO"
10 A=123
15 B=678
20 C$="CORTO"
25 PRINT"A=";A,"B=";B
26 PRINT"C$=";C$
27 PRINT#4:CLOSE4
30 LOAD"LUNGO",8
```

```
1 REM LUNGO
5 OPEN4,4:CMD4:PRINT"RUN LUNGO"
10 D=998
15 E=790
20 F$="LUNGO"
25 PRINT"A=";A,"B=";B
26 PRINT"C$=";C$
28 PRINT"D=";D,"E=";E
29 PRINT"F$=";F$
30 PRINT#4:CLOSE4
40 STOP
50 END
```

RISULTATI PROGRAMMI PRECEDENTI

```
RUN CORTO
A= 123            B= 678
C$=CORTO


RUN LUNGO
A= 0            B= 0
C$=
D= 998            E= 790
F$=LUNGO
```

1.6 PROGRAMMI E SUGGERIMENTI UTILI

Prima di abbandonare l'argomento toccato nel precedente paragrafo, desideriamo suggerirti un metodo da adottare quando si provano programmi complessi, nei quali sono probabilmente contenuti errori.
Questa implementazione del Basic quando si interrompe un programma e si fanno delle modifiche o si entra in EDITOR, cancella tutte le variabili già in memoria. Per evitare questo fatto, che a volte risulta fastidioso, basta, dopo aver caricato il programma, leggere in modo immediato il contenuto dei byte 45 e 46 (puntatori inizio variabili), aggiungergli qualcosa (per un migliaio di byte basta aggiungere 4 al byte alto) e riscrivere con PEEK i nuovi valori nelle 3 coppie di puntatori (45/46, 47/48, 49/50). Dopo di ciò si può dare RUN al programma, interromperlo, andare a modificare le istruzioni (basta non allungare troppo per non invadere la zona variabili) senza perdere le variabili. Esse si perdono se dai ancora RUN, ma puoi ripartire con GOTO e portare avanti rapidamente le tue prove. Ricordati però di rimettere a posto le cose dopo.

Un altro utile suggerimento è quello di sfruttare le possibilità offerte dall'EDITOR per fare poca fatica a scrivere i programmi. Per esempio tutte le linee simili si possono scrivere una dopo l'altra, muovendosi con il cursore e andando a modificare il numero di linea e le parti variate. Attenzione però con le stringhe tra apici, sono un pò delicate da trattare.

Spesso risulta utile fondere dei programmi già esistenti, ma non sempre disponiamo del TOOL KIT che ci servirebbe. Vediamo come si fa ad ottenere la fusione dei programmi (MERGE), con qualche limitazione però. Posssiamo suggerirti due metodi.
Il primo consente di aggiungere a un programma uno o più programmi che però abbiano numeri di linea maggiori in sequenza. Si procede così:
.. carichiamo in memoria il programma con i numeri di linea più bassi;
.. eseguiamo in immediato: PRINT PEEK(45),PEEK(46), ottenendo i byte LO e HI del puntatore all'inizio delle variabili;
.. il numero ottenuto diminuito di 2 unità è l'indirizzo dell'ultimo LINK del programma presente in memoria (i 2 byte a zero che segnalano la fine del programma);
.. scriviamo con POKE i valori dei due byte LO e HI ottenuti nei byte 43 e 44, ottenendo di spostare l'inizio del programma Basic all'indirizzo del LINK nullo precedente;
.. carichiamo con LOAD il programma successivo, quello con i numeri di linea maggiori;
.. ripetiamo la procedura di lettura dei byte 45 e 46 e di riscrittura dei byte 43 e 44 prima descritta fino ad aver caricato tutta la catena dei programmi;
.. alla fine riscriviamo nei byte 43 e 44 i valori iniziali;

.. eseguiamo LIST e vediamo tutto il nostro programma, che può essere memorizzato o su disco o su nastro.

Il secondo metodo consente di fondere programmi anche con numeri di linea che si intrecciano, ma è un pò macchinoso. Supponiamo di voler fondere il programma PRG1 con il programma PRG2. Prendiamo il programmma più corto (PRG2) e lo memorizziamo a pezzi abbastanza corti da poter stare listati su due terzi del video, assegnando ad ogni pezzo un nome. Poi carichiamo il primo pezzo e lo listiamo sul video, carichiamo PRG1, ci portiamo con il cursore sul video e facciamo entrare le linee listate in PRG1 premendo RETURN su ogni linea. A questo punto PRG1 contiene anche un pezzo di PRG2; dobbiamo memorizzarlo, poi richiamare il secondo pezzo di PRG2 e ricominciare. Se sei disposto alla pazienza puoi anche sperimentare questo metodo.

Vediamo ora come si possono cancellare blocchi di linee di programma usando una semplice routine RDEL (cosa necessaria per relizzare il MERGE con il secondo metodo, sopra esposto).

```
50000 REM ROUTINE DELETE
50001 INPUT"DA, A, PASSO";DD,AD,PD
50002 PRINTCHR$(147)
50003 PRINTCHR$(19)DD
50004 DD=DD+PD
50005 PRINT"50010 DD="DD":AD="AD":PD="PD
50006 PRINT"GOTO50010"
50007 POKE631,19:POKE632,13:POKE633,13
50008 POKE634,13:POKE198,4:END
50010 DD= 81::AD= 80::PD= 1
50011 IFDD>ADTHENPRINTCHR$(147):END
50012 GOTO50003
```

Questa routine va aggiunta al programma da modificare con cancellazioni di blocchi di linee. Essa puo essere rilocata cambiando i numeri di linea. Devi farla partire scrivendo GOTO50000. Vediamo cosa succede:
.. alla 50001 viene chiesto da quale linea a quale linea e con quale incremento cancellare;
.. alla 50002 viene eseguito CLR/HOME;
.. alla 50003 viene eseguito HOME e scritto il numero di linea iniziale da cancellare, DD;
.. alla 50004 si incrementa DD con il passo PD;
.. alla 50005 si scrive sul video la linea di programma 50010 che pone le variabili DD, AD e PD ai loro valori assegnati;

.. alla 50006 si scrive sotto sul video GOTO50010;
..alla 50007 si pongono dei caratteri nel buffer della tastiera: HOME, RETURN, RETURN;
.. alla 50008 si pone ancora nel buffer della tastiera un carattere RETURN e si pone il contatore dei caratteri del buffer della tastiera al valore 4 (4 caratteri);
.. il buffer della tastiera si svuota automaticamente eseguendo con il primo RETURN la cancellazione del numero di linea scritto in cima al video, infatti il cursore viene mandato in alto dal carattere HOME che precede il RETURN; gli altri due RETURN fanno accettare la linea 50010 modificata e la fanno eseguire per effetto del GOTO50010;
.. alla 50011 viene controllato che DD non abbia raggiunto il limite AD, nel caso il programma termina;
.. alla 50012 si ha GOTO50003 e si ricomincia.

In questa routine si forza da programma l'intervento dell'EDITOR servendosi del buffer della tastiera.

CAPITOLO 2

I FILE SU STAMPANTE

2.1 INTRODUZIONE

In questo capitolo descriviamo l'uso della stampante COMMODORE MPS-803, che viene venduta per il calcolatore COMMODORE 64. Il calcolatore può essere collegato anche ad altre stampanti senza problemi; alcune si possono collegare direttamente, per altre è necessaria un'interfaccia speciale. La maggior parte dei discorsi che facciamo restano validi anche per altri tipi di stampanti; nel caso devi chiederne al venditore le caratteristiche, vedere se è necessaria un'interfaccia e leggere accuratamente il manuale allegato per scoprire le differenze.

Figura 2.1 Stampante MPS-803

La MPS-803 è una stampante a impatto a matrice di punti, nella quale un carattere è formato da 6 punti orizzontali e 7 punti verticali. Essa riconosce e stampa tutti i caratteri dei due set disponibili sul COMMODORE 64: maiuscolo/grafico e minuscolo/maiuscolo. Inoltre può stampare colonne di punti (7 punti verticali) e quindi è una stampante grafica; la colonna di punti deve essere opportunamente codificata.
Essa si collega tramite l'interfaccia seriale standard al COMMODORE 64, mediante il cavo fornito che termina con uno spinotto DIN a 6-pin, dove i pin hanno il significato riportato nella Figura 2.2.

CONNETTORE

Pin No.	Signal
1	N.C.
2	GND
3	SERIAL ATN
4	SERIAL CLK
5	SERIAL DATA
6	RESET

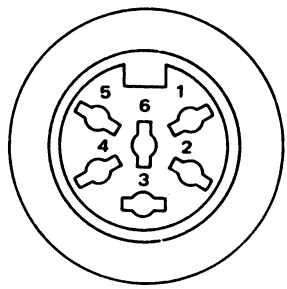


Figura 2.2 Presa di collegamento per la MPS-803

Sulla parte superiore della MPS-803 sono visibili:
..in alto a sinistra, la manopola per l'avanzamento manuale della carta;
..in alto a destra, la levetta per il bloccaggio della carta (da usare quando non è montato il trascina moduli);
..in basso a destra, una spia luminosa rossa, marcata Power, che segnala:
.con luce continua che la macchina è accesa,
.con luce intermittente che si è verificato un errore,
e vicino un tasto a pressione, marcato Paper Advance, per l'avanzamento della carta.
La copertura anteriore può essere sollevata, agendo su due appositi incavi laterali. Per inserire il nastro inchiostrato si deve sollevare tale copertura e si rende visibile l'alloggiamento del nastro. Nel manuale allegato alla stampante sono riportate delle illustrazioni che insegnano a montare il nastro.
Sul lato destro in basso si trova l'interruttore di accensione.

La stampante può essere alimentata con moduli continui o con fogli singoli, usando la levetta di bloccaggio della carta (quando è in posizione OPEN la carta è libera);

oppure può essere montato il trascina moduli, richiedendolo al venditore. La larghezza della linea di stampa è di 80 caratteri (larghi 6 punti), quindi di 480 punti. Possiamo assumere la larghezza della linea di stampa come dimensione del record fisico. La stampa è bidirezionale e la velocità di stampa è di 60 caratteri al secondo. Si possono stampare fino a 3 copie. La stampante non può funzionare se la carta non è inserita. Quando, durante la stampa, termina la carta, comincia a pulsare la spia rossa Power e la stampa si interrompe; per proseguire devi inserire nuova carta e premere il tasto Paper Advance.

Nella parte posteriore vediamo quanto riportato nella Figura 2.3.

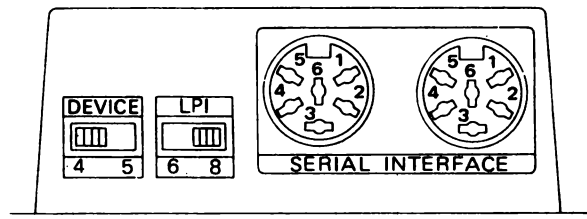


Figura 2.3 Parte posteriore della stampante MPS-803

Lo switch DEVICE può essere posto nella posizione 4 o nella posizione 5, dando la possibilità di scegliere il "dn" della stampante. Di solito si usa il 4, ma, se sono collegate contemporaneamente due stampanti, una deve avere dn=5.
Lo switch LPI può essere posto nelle posizioni 6 o 8; esso indica la distanza di 1/6 o 1/8 di pollice tra due linee di stampa.
La porta seriale SERIAL INTERFACE dispone di due ingressi; uno serve per collegare la stampante al calcolatore, l'altro per collegare in cascata una seconda stampante o una unità a floppy disk.
Il COMMODORE 64 può inviare dati alla stampante dopo avere stabilito una comunicazione con essa; la stampante riceve un flusso di dati e stampa un file. La MPS-803 contiene un microprocessore che gestisce le operazioni di stampa, eseguendo apposite routine memorizzate in una ROM interna, servendosi di una descrizione dei caratteri memorizzata in una ROM interna, e di un buffer di stampa, cioè di una parte di RAM interna.
Il COMMODORE 64 invia i dati, attraverso l'interfaccia seriale, un bit dopo l'altro; ogni gruppo di 8 bit (un byte) contiene un codice ASCII, che può variare da

0 a 255. I byte sono ricevuti e memorizzati uno dopo l'altro nel buffer, che può contenere fino a 90 byte; vedremo più avanti in quali circostanze avviene realmente la stampa.
Alcuni dei codici inviati non sono caratteri da stampare, ma codici di controllo che agiscono sul modo di funzionare della stampante.
Quando la stampante funziona, al momento dell'accensione, sia che sia collegata al calcolatore, sia che non lo sia, la testina di stampa viene spostata dal centro verso sinistra e poi riportata al centro.
Per vedere se la stampa è corretta puoi farle eseguire l'auto-test. Devi procedere così:
..aver montato correttamente il nastro e inserito la carta,
..dare corrente alla stampante, anche non collegata al calcolatore, mentre tieni premuto il tasto anteriore "Paper Advance", poi rilasciare tale tasto.
La stampante stampa ripetutamente tutti i caratteri stampabili. Per interrompere il test devi togliere corrente.

2.2 ISTRUZIONI DI STAMPA

Prima di stampare si deve aprire la comunicazione tra il calcolatore e la stampante con l'istruzione OPEN; essa si scrive:

OPEN lfn, dn, sa

..lfn (Logical File Number), è il numero logico del file che si vuole aprire e che deve essere usato nelle istruzioni di stampa successive; esso può variare da 0 a 255. Se lfn>127 si ottiene una spaziatura doppia tra una linea e la successiva.
..dn (Device Number), è il numero della periferica, può essere 4 o 5, e deve essere quello predisposto dall'apposito switch posto sul retro della stampante.
..sa (Secondary Address), è un numero che può valere 0 (valore di default) o 7 e determina il set di caratteri che la stampante deve usare nelle successive operazioni di stampa:
.0, o niente, per il set maiuscolo/grafico,
.7, per il set minuscolo/maiuscolo.
I parametri possono essere costanti o variabili con valore intero.
Il set di caratteri scelto con la OPEN resta attivo fino alla CLOSE; si può usare all'interno della lista di stampa un codice di controllo che modifica temporaneamente il set, ma esso è valido solo per la stampa in corso, cioè fino al primo carattere RETURN.

L'istruzione di stampa è:

PRINT# lfn, lista

..lfn, deve essere lo stesso usato nella OPEN.
..lista, è l'insieme dei dati da stampare, dei caratteri separatori, delle funzioni di stampa e dei codici di controllo per la stampante. Negli esempi successivi vedremo le caratteristiche di ogni elemento che può comparire nella lista.
Per scrivere questa istruzione non si può usare il "?" per abbreviare la parola chiave, inoltre non deve esserci spazio prima del carattere "#".
Il calcolatore, dopo aver eseguito un'istruzione PRINT#, chiude la comunicazione con la stampante, ma il file logico rimane aperto; esso viene chiuso solo dall'istruzione CLOSE.

Si può stampare su carta, anche usando l'istruzione CMD, dopo aver aperto un file per la stampante. Questa istruzione trasferisce l'uscita, che normalmente avviene sul video, alla stampante. Si scrive:

```
CMD lfn, lista
```

..lfn, deve essere lo stesso usato nell'istruzione OPEN.
..lista, può essere una normale lista di stampa o mancare.

Dopo l'esecuzione di CMD, con o senza "lista", le successive istruzioni PRINT (scritte senza il carattere "#") mandano l'output sulla stampante. Per far terminare l'effetto di CMD, cioè il "dirottamento" dell'uscita dal video a un'altra periferica, è necessario eseguire una PRINT#lfn, con "lfn" uguale a quello usato per CMD, che serve per chiudere la linea, prima della CLOSE. In caso contrario non si ha un funzionamento corretto del sistema; inoltre, quando si è in stato CMD, non si deve accedere a un'altra periferica collegata in serie (disco o altra stampante). Il comando LIST, usato dopo CMD, provoca la lista del programma sulla stampante.
Il modo corretto per ottenere la lista dei programmi sulla stampante è eseguire in immediato le due serie di comandi:

```
OPEN4,4:CMD4:LIST
PRINT#4:CLOSE4
```

Le serie di comandi sono due perché dopo LIST non viene restituito il controllo al BASIC.

Quando le operazioni di stampa sono terminate, deve essere eseguita l'istruzione:

```
CLOSE lfn
```

che serve per chiudere la comunicazione. Dopo l'esecuzione della CLOSE non si può più comunicare con il file "lfn", se non si esegue nuovamente una OPEN.

È importante non lasciare aperti i file che non servono più; si rischia di occupare inutilmente posto nella tabella dei file, che ha solo 10 posti.

Le istruzioni di stampa possono essere usate sia in modo immediato che da programma.

ESEMPI DI STAMPA IN MODO IMMEDIATO

Usando in modo immediato le 4 sequenze di istruzioni che seguono, si ottiene sempre lo stesso risultato sulla stampante e si opera correttamente chiudendo sia la linea che il file logico.

1) OPEN4, 4:PRINT#4, "MPS-803":CLOSE4
   stampa MPS-803 e va a capo.
2) OPEN4, 4:CMD4, "MPS-803":PRINT#4:CLOSE4
   stampa MPS-803 e va a capo.
3) OPEN4, 4:CMD4:PRINT"MPS-803":PRINT#4:CLOSE4
   stampa MPS-803 e va a capo.
4) OPEN4, 4, 7:CMD4:PRINT"MPS-803":PRINT#4:CLOSE4
   stampa mps-803 (abbiamo usato sa=7) e va a capo.

ESEMPI DI STAMPA DA PROGRAMMA

Il programma SA0CMDLIST, che segue con i suoi risultati, apre il file logico con lfn=1 sulla stampante (dn=4), usando sa=0, che poteva anche essere saltato, dato che 0 è il valore di default, e quindi lavora con il set maiuscolo/grafico. Nel programma alla linea 20 è inserito il comando LIST, cioè il programma stampa una frase e poi lista se stesso.
Dopo l'esecuzione del programma devi eseguire in immediato:

```
OPEN CON SA=0 E USO CMD E LIST

1 REM SA0CMDLIST
5 OPEN1,4,0:REM APRE CON SA=0
10 CMD1:REM USCITA DA VIDEO A STAMPANTE
15 PRINT"OPEN CON SA=0 E USO CMD E LIST"
20 PRINT:LIST
```

Il programma SA7CMDLIST è uguale al precedente, salvo che apre il file di stampa con sa=7 e quindi lavora con il set di caratteri minuscolo/maiuscolo.

```
open con sa=7 e uso cmd e list

1 rem sa7cmdlist
5 open1,4,7:rem apre con sa=7
10 cmd1:rem uscita da video a stampante
15 print"open con sa=7 e uso cmd e list"
20 print:list
```

Segue il programma SETMPS-803, che stampa in forma tabellare i due set di caratteri disponibili sulla stampante. Il disegno dei caratteri è quello registrato nella ROM interna alla stampante, che differisce un po' dalla forma dei caratteri che appare sul video: 6x7 punti per la stampante, 8x8 punti sul video.

```
1 REM SETMPS-803
5 A$="      SET MAIUSCOLO/GRAFICO"
6 SA=0
10 OPEN4,4,SA:PRINT#4,A$
11 PRINT#4:PRINT#4," | ";
12 FORK=0TO9:PRINT#4,MID$(STR$(K),2);" ";
13 NEXTK
14 FORK=65TO70:PRINT#4,CHR$(K);" ";:NEXTK
15 PRINT#4:PRINT#4,CHR$(192)"+";
16 FORK=1TO16:PRINT#4,CHR$(192)CHR$(192);
17 NEXTK:PRINT#4:I=0
19 FORK=0TO9:PRINT#4,MID$(STR$(K),2);"| ";
20 GOSUB100:I=I+1:NEXTK
23 FORK=65TO70:PRINT#4,CHR$(K);"| ";
25 GOSUB100:I=I+1:NEXTK
27 IFSA=7THENCLOSE4:STOP
29 PRINT#4:PRINT#4:SA=7
30 A$="      SET MAIUSCOLO/MINUSCOLO"
31 CLOSE4:GOTO10
100 IK=I:FORL=1TO16
101 IFIK<32THENPRINT#4,"  ";:GOTO104
102 IFIK>127ANDIK<160THENPRINT#4,"  ";:GOTO104
103 PRINT#4,CHR$(IK);" ";
104 IK=IK+16:NEXTL:PRINT#4:RETURN
```

I caratteri da stampare sono inviati come codici ASCII; il programma stampa spazi al posto dei caratteri corrispondenti ai codici da 0 a 31 e da 128 a 159, formando le due colonne vuote nelle tabelle, dato che non corrispondono a caratteri stampabili. Le coordinate orizzontali e verticali delle due tabelle sono espresse in esadecimale (le cifre esadecimali da A e F corrispondono ai numeri decimali da 10 a 15).

Per ottenere il codice ASCII dei caratteri devi moltiplicare per 16 la coordinata di colonna (sopra) e aggiungere la coordinata di riga (laterale).

RISULTATI PROGRAMMA SETMPS-803

SET MAIUSCOLO/GRAFICO

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	—	┐				┌	—	┐		┌
1			!	1	A	Q	♠	●			▮	┴	♠	●	▮	┴
2			"	2	B	R	│	─			▬	┬	│	─	▬	┬
3			#	3	C	S	─	♥			═	┼	─	♥	═	┼
4			$	4	D	T	─	│			_	│	─	│	_	│
5			%	5	E	U	─	╱			▏	▌	─	╱	▏	▌
6			&	6	F	V	─	╳			▒	▐	─	╳	▒	▐
7			'	7	G	W	│	O			│	─	│	O	│	─
8			(	8	H	X	│	♠			▒	─	│	♠	▒	─
9			)	9	I	Y	╲	│			◤	▬	╲	│	◤	▬
A			*	:	J	Z	╲	◆			│	┘	╲	◆	│	┘
B			+	;	K	[	╱	+			├	▪	╱	+	├	▪
C			,	<	L	£	L	▒			▪	▪	L	▒	▪	▪
D			-	=	M	]	╲	│			└	┘	╲	│	└	┘
E			.	>	N	↑	╱	π			┐	▪	╱	π	┐	▪
F			/	?	O	←	┌	◥			▁	▝	┌	◥	▁	π

set maiuscolo/minuscolo

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0				0	@	P	—	P				┌	—	P		┌
1			!	1	a	q	A	Q			▮	┴	A	Q	▮	┴
2			"	2	b	r	B	R			▬	┬	B	R	▬	┬
3			#	3	c	s	C	S			═	┼	C	S	═	┼
4			$	4	d	t	D	T			_	│	D	T	_	│
5			%	5	e	u	E	U			▏	▌	E	U	▏	▌
6			&	6	f	v	F	V			▒	▐	F	V	▒	▐
7			'	7	g	w	G	W			│	─	G	W	│	─
8			(	8	h	x	H	X			▒	─	H	X	▒	─
9			)	9	i	y	I	Y			▨	▬	I	Y	▨	▬
a			*	:	j	z	J	Z			│	▼	J	Z	│	▼
b			+	;	k	[	K	+			├	▪	K	+	├	▪
c			,	<	l	£	L	▒			▪	▪	L	▒	▪	▪
d			-	=	m	]	M	│			└	┘	M	│	└	┘
e			.	>	n	↑	N	╳			┐	▪	N	╳	┐	▪
f			/	?	o	←	O	▨			▁	▝	O	▨	▁	╳

COMMENTO A SETMPS-803

..5/6: prepara in A$ l'intestazione della prima tabella e pone SA=0.

..10: apre il file logico con lfn=4, il valore attuale di SA per la stampante, e stampa il titolo della tabellina.

..11/17: stampa le coordinate di colonna e le linee separatrici (CHR$(192) dà luogo a un trattino), e pone il contatore I a 0.
..19/20: stampa le linee corrispondenti alle coordinate di riga da 0 a 9, servendosi del sottoprogramma in 100 per stampare i caratteri del set attivo.
..23/25: come sopra per le linee con coordinate da A a F.
..27: se SA=7 chiude il file logico e si ferma.
..29/31: pone SA=7, prepara la nuova intestazione in A$, chiude il file logico e torna alla linea 10 per stampare la seconda tabellina.
..100/104: sottoprogramma che in base al codice del carattere, che trova inizialmente in I e pone in IK, stampa, se sono stampabili, i 16 caratteri di una linea. Non stampa il carattere se il codice è minore di 32 oppure compreso tra 128 e 159, estremi inclusi.
I caratteri che stanno su una linea hanno codici che differiscono di 16 da quello precedente.

2.3 MODI DI STAMPA E USO DEI CARATTERI DI CONTROLLO

Il valore di "sa" usato nella OPEN predispone l'uso di uno dei due set di caratteri disponibili nella MPS-803, in modo permanente, fino all'esecuzione della relativa CLOSE. Il valore di "sa" influenza la stampa in MODO CARATTERE, cioè quella attiva al momento dell'accensione della stampante. Per ottenere il passaggio al MODO GRAFICO devi inviare un carattere di controllo (CHR$(8)); in tale caso è necessario inviare un altro carattere di controllo per tornare al modo testo (CHR$(15)).
Possiamo considerare come una prima distinzione nelle possibilità di stampa i due modi: carattere e grafico; ad essi possiamo aggiungere gli altri modi che si ottengono con l'uso di alcuni caratteri di controllo.
Segue l'elenco dei caratteri di controllo che hanno un particolare significato per la stampante MPS-803; essi devono essere inviati nella "lista" di un'istruzione PRINT diretta alla stampante, come stringa, usando la funzione CHR$. Per ogni carattere indichiamo: il codice, il significato, i parametri che devono seguire il codice, se necessario, e il numero dei byte che vengono occupati in conseguenza nel buffer della stampante. Se un codice richiede dei parametri, ovviamente, questi vengono trasmessi, ma non stampati. Alcuni codici vanno usati insieme ad altri, altrimenti perdono significato.
Tieni presente che i codici di controllo hanno in generale significato diverso se diretti al video invece che alla stampante; gli unici che producono lo stesso effetto sono 10, 13, 18 e 146.

CODICE	SIGNIFICATO	PARAMETRI	NUM. BYTE
8	MODO GRAFICO	no	1
10	invio LINE FEED e RETURN	no	1
13	invio LINE FEED e RETURN	no	1
14	MODO CARATTERE ALLARGATO	no	1
15	MODO CARATTERE NORMALE	no	1
16	SPOSTAMENTO POSIZIONE STAMPA A UNA COLONNA	2	3
17	PASSAGGIO SET MINUSC./MAIUSC.	no	1
18	MODO RVS-ON	no	1
26	MODO RIPETIZIONE CAR. GRAFICO	1	2
27	SPOSTAMENTO POSIZIONE GRAFICA deve essere seguito da CHR$(16)	2	4
145	PASSAGGIO SET MAIUSC./GRAFICO	no	1
146	MODO RVS-OFF	no	1

Esaminiamo dettagliatamente i singoli codici di controllo, riportando alcuni programmi esempio. In alcuni programmi esempio abbiamo usato la tecnica di far lavorare il programma stampando i risultati, poi il programma lista se stesso; in conseguenza, in questi casi, vedrai prima i risultati e poi il listato del programma.

CHR$(8) MODO GRAFICO

Predispone la stampa in modo grafico; tale modo resta attivo fino all'invio dei codici di controllo 14 o 15, che riportano rispettivamente in modo carattere allargato e in modo carattere normale.
Dopo aver attivato il modo grafico devi passare nella lista di stampa i codici dei caratteri grafici da stampare, come stringa. Ogni carattere grafico è formato da una colonna di 7 punti, e viene codificato secondo le seguenti regole:
..i punti da disegnare devono corrispondere alla cifra 1, quelli da non disegnare devono corrispondere alla cifra 0,

..le 7 linee hanno pesi diversi, partendo dall'alto i pesi sono: 1, 2, 4, 8, 16, 32, 64, cioè le potenze di 2, partendo dall'esponente 0 e arrivando all'esponente 6,
..devi moltiplicare ogni cifra per il peso corrispondente e sommare i valori, per una colonna con 7 cifre 1 ottieni 1+2+4+8+16+32+64=127, per una colonna con 7 cifre 0 ottieni 0.
..al numero ottenuto devi aggiungere 128, in tale modo il codice calcolato può variare da 128 a 255.
Il codice deve essere passato nella "lista" di stampa con la funzione CHR$. Se desideri ottenere un disegno composto da più colonne, devi ripetere il procedimento spiegato per ogni colonna.
Abbiamo preparato il disegno di un omino con le braccia alzate, occupando 11 colonne di 7 punti; lo riportiamo indicando con asterischi i punti da disegnare e con lineette quelli da lasciare in bianco. Inoltre riportiamo di fianco l'immagine ottenuta con 0 e 1 e con a margine i pesi di ogni punto e sotto ogni colonna il codice risultante dal calcolo.

- * * - * * * - * * -	1)	0	1	1	0	1	1	1	0	1	1	0
- - * * - * - * * - -	2)	0	0	1	1	0	1	0	1	1	0	0
- - - * * * * * - - -	4)	0	0	0	1	1	1	1	1	0	0	0
- - - - * * * - - - -	8)	0	0	0	0	1	1	1	0	0	0	0
- - - - * * * - - - -	16)	0	0	0	0	1	1	1	0	0	0	0
- - - * * - * * - - -	32)	0	0	0	1	1	0	1	1	0	0	0
- * * * - - - * * * -	64)	0	1	1	1	0	0	0	1	1	1	0
Valore codici:		0	65	67	102	61	31	61	102	67	65	0

Aggiungendo 128 agli 11 codici otteniamo:
128, 193, 195, 230, 189, 159, 189, 230, 195, 193, 128, passando dopo CHR$(8) le funzioni CHR$ degli 11 codici otteniamo il disegno di un omino. Il nostro disegno è simmetrico e inizia e finisce con una colonna bianca; per questo disegnando vicino alcuni omini essi non risultano attaccati insieme.

Nel programma COD8-1 abbiamo inserito gli 11 numeri con una linea DATA, la 4; poi prepariamo nella stringa A$ l'omino completo come stringa; stampando A$, in modo grafico, otteniamo il disegno.

```
XXXXX
XXXXXXX
XXXXXXXXX
XXXXXXXXXXX
```

```
1 rem cod8-1
2 open10,4,7
3 cmd10
4 data0,65,67,102,61,31,61,102,67,65,0
5 a$="":forj=1to11
6 reada:a=a+128
7 a$=a$+chr$(a)
8 nextj
9 fork=1to5
10 printchr$(8)a$;
11 nextk:print:print
12 fork=1to7
13 printa$;
14 nextk:print:print
15 fork=1to9
16 printa$;
17 nextk:print:print
18 fork=1to11
19 printa$;
20 nextk:print
21 printchr$(15)
22 print:list
```

Dopo l'esecuzione del programma devi eseguire in modo immediato:

PRINT#10:CLOSE 10.

COMMENTO A COD8-1

..2/3: apre la stampante con lfn=10 e sa=7, con CMD10 trasferisce l'uscita video alla stampante.

..4: linea DATA con i codici delle 11 colonne di punti, prima di aggiungere 128.

..5/8: legge gli 11 codici, aggiunge 128 a ogni codice e costruisce la stringa A$ con il disegno.

..9/11: stampa 5 omini, dopo aver attivato la stampa grafica con CHR$(8). Il codice è usato all'interno del ciclo FOR, e quindi viene inviato 5 volte; bastava inviarlo una sola volta prima del ciclo FOR. Alla linea 11 esegue 2 volte PRINT, la prima volta per andare a capo e la seconda per produrre uno spazio. In modo grafico la distanza tra le righe è minore rispetto al modo carattere.

..12/14: stampa 7 omini, lavorando come sopra, ma senza inviare di nuovo il codice 8.

..15/17: stampa 9 omini.

..18/20: stampa 11 omini.

..21: ritorna in modo carattere normale stampando CHR$(15)

..22: lista se stesso.

Nel programma esempio COD8-2, abbiamo disegnato una casetta che occupa un rettangolo di 17 colonne e 21 righe. Per poterla stampare la dobbiamo dividere a strisce alte 7 punti ciascuna; otteniamo 3 strisce. Stamperemo le tre parti una per riga, una sotto l'altra.

Per evitare la fatica di calcolare i codici abbiamo disegnato la casetta in 21 linee di programma, 21 linee DATA, scritte una dopo l'altra. Prima, con una REM abbiamo numerato le colonne, per disegnare più facilmente. La casetta viene disegnata usando gli asterischi.
Abbiamo incorporato nel programma una routine che legge le 21 stringhe, 7 per volta, analizza i caratteri delle stringhe e sostituisce 1 agli asterischi e 0 agli spazi. Poi calcola il valore di ogni colonna, aggiunge 128 e costruisce la stringa con la funzione CHR$ di ogni codice.
Puoi estrarre questa routine e adattarla a disegni di altre dimensioni. Ricordati di usare un numero di righe multiplo di 7.
Il programma stampa 10 volte la casetta e poi lista se stesso.

```
1 REM COD8-2
3 DIMT$(7),C$(3):REM STRINGHE PER DISEGNO
4 DIMT(6):REM PER CALCOLO
8 REM DISEGNO CASETTA IN 21 LINEE DATA
9 REM   1234567890123456 7
10 DATA"                 "
11 DATA"                 * "
12 DATA"                 * "
13 DATA"          *      * "
14 DATA"        * *    *   "
15 DATA"       *    * ***  "
16 DATA"      *      ** *  "
17 DATA"     *        * *  "
18 DATA"    *          **  "
19 DATA"   *            *  "
20 DATA" ***************  "
21 DATA" *               * "
22 DATA" * ***      *** * "
23 DATA" * * *      * * * "
24 DATA" * ***      *** * "
25 DATA" *               * "
26 DATA" *               * "
27 DATA" *      ***      * "
28 DATA" *      * *      * "
29 DATA" *      * *      * "
30 DATA" ***************  "
36 PRINT"□STO CALCOLANDO I CODICI DELLA CASETTA"
40 REM CALCOLO COLONNE DI PUNTI
45 FORK=1TO3:C$(K)="":FORJ=1TO7
50 READT$(J):PRINTT$(J):NEXTJ
55 FORL=1TO17:I=0
60 FORJ=1TO7:T=ASC(MID$(T$(J),L,1))
65 T(I)=0:IFT=42THENT(I)=1
70 I=I+1:NEXTJ
75 T=0:FORJ=0TO6:T=T+T(J)*2↑J:NEXTJ:T=T+128
80 C$(K)=C$(K)+CHR$(T):NEXTL
85 NEXTK
100 REM STAMPA CASETTA
105 OPEN4,4:PRINT#4,CHR$(8)
110 FORJ=1TO3:FORK=1TO10
115 PRINT#4,C$(J);:NEXTK:PRINT#4
120 NEXTJ
125 PRINT#4,CHR$(15)
130 CMD4:LIST
```

Dopo l'esecuzione del programma devi eseguire in modo immediato:

PRINT#4:CLOSE4.

COMMENTO A COD8-2
..3: dimensiona T$(7) per contenere le 7 stringhe di una striscia, e C$(3) per contenere le 3 stringhe da stampare per formare il disegno.
..4: dimensiona un vettore T(6) per contenere i 7 numeri corrispondenti a una colonna di punti.
..8/30: disegno della casetta in linee DATA.
..36: stampa un messaggio per avvisare che esegue il calcolo dei codici.
..40/85: ciclo per calcolare i codici della casetta.

.45: inizia il ciclo per K da 1 a 3 per calcolare le 3 stringhe che costruiscono il disegno, pulisce la stringa e inizia un ciclo per J da 1 a 7 per leggere 7 linee DATA.

.50: legge la stringa, la stampa sul video, disegnando così la casetta con gli asterischi, e chiude il ciclo di J.

.55: inizia il ciclo per analizzare i 17 caratteri di ogni stringa, e pone I=0, indice per il vettore T.

.60/70: preleva da ognuna delle 7 stringhe i 7 caratteri che occupano la stessa posizione verticale e trasforma gli asterischi in 1 e gli spazi in 0 prima di porli nel vettore T(I). Alla fine del ciclo il vettore T(I) contiene una colonna di punti in cifre 1 e 0.

.75: calcola il codice corrispondente moltiplicando le cifre per i relativi pesi, poi aggiunge 128.

.80: aggiunge alla stringa C$(K) il nuovo carattere grafico calcolato e chiude il ciclo di L. Alla fine di questo ciclo la stringa C$(K) contiene tutti i 17 codici della striscia relativa, trasformati in carattere ASCII.

.85: chiude il ciclo di K. Alla fine sono pronte le 3 stringhe C$(K).
..100/120: stampa 10 casette ripetendo 10 volte ogni disegno e poi andando a capo.
..125: disattiva il modo grafico e torna al modo carattere normale.
..130: lista se stesso.

CHR$(10) e CHR$(13) INVIO LINE FEED E RETURN

Questi due codici hanno lo stesso comportamento, ognuno di essi stampa un LINE FEED e un RETURN e provoca la stampa di tutto quello che è contenuto nel buffer. Se termini una "lista" con uno di questi codici, e non aggiungi il ";" finale, la mancanza di punteggiatura provoca un ulteriore RETURN.
Nel programma COD10/13 che segue, ti mostriamo come l'effetto dei due codici sia il medesimo e come influisce il valore di "lfn" sulla spaziatura tra le linee.

```
1 REM COD10/13
5 OPEN129,4
10 A$="PROVA1 LFN>128":B$="PROVA DI CHR$(10)"
15 PRINT#129,A$CHR$(10)B$
20 C$="PROVA2 LFN>128":D$="PROVA DI CHR$(13)"
25 PRINT#129,C$CHR$(13)D$
30 CLOSE129
35 OPEN10,4
40 E$="PROVA3 LFN<128":F$="PROVA DI CHR$(10)"
45 PRINT#10,E$CHR$(10)F$
50 G$="PROVA4 LFN<128":H$="PROVA DI CHR$(13)"
55 PRINT#10,G$CHR$(13)H$
60 CLOSE10
65 STOP
```

RISULTATI PROGRAMMA COD10/13

```
PROVA1 LFN>128
PROVA DI CHR$(10)

PROVA2 LFN>128
PROVA DI CHR$(13)

PROVA3 LFN<128
PROVA DI CHR$(10)
PROVA4 LFN<128
PROVA DI CHR$(13)
```

COMMENTO A COD10/13

..5: apre con lfn=129 (>127) e questo provoca una doppia spaziatura a fine linea se manca la punteggiatura finale.

..10/15: prepara le due stringhe A$ e B$ e le stampa separandole con CHR$(10); esse vengono stampate una su ogni riga, ma alla fine si ha un doppio spazio.

..20/25: come sopra, ma separando le due stringhe con CHR$(13); si ottiene lo stesso effetto di prima.

..30: chiude il file con lfn=129.

..35: apre il file con lfn=10 (<127) e questo provoca una spaziatura semplice in assenza di punteggiatura finale.

..40/60: stampa con gli stessi codici di controllo e ottiene la spaziatura semplice anche in assenza di punteggiatura finale.

Se la punteggiatura finale invece di essere un ";" è una ",", si ha l'aggiunta di 10 spazi, e questo può provocare il passaggio a nuova linea, anche se non sono presenti o il codice 10 o il codice 13.

CHR$(14) MODO CARATTERE ALLARGATO

Predispone la stampa in modo testo del carattere allargato, cioè del carattere formato da 12 punti per riga e 7 punti per colonna. La predisposizione rimane fino a quando si invia il codice 15, per tornare al carattere normale, o il codice 8 per passare in modo grafico.
Segue il programma COD14-1, che stampa due linee in carattere allargato, poi torna al modo normale e lista se stesso.
Dopo l'esecuzione del programma devi eseguire in modo immediato:

PRINT#10=CLOSE10

```
COMMODORE
  MPS-803

1 REM COD14-1
2 OPEN10,4
3 CMD10
4 PRINTCHR$(14)"COMMODORE"
5 PRINTCHR$(14)" MPS-803 "
6 PRINTCHR$(15)
7 LIST
8 PRINT#10:CLOSE10
```

Il programma COD14-2, invece, mostra come si possono ottenere sulle stesse linee di stampa sia caratteri normali che caratteri allargati, usando alternativamente i codici 14 e 15.

```
1 REM COD14-2
2 OPEN10,4
3 PRINT#10,CHR$(14)"COMMODORE ";
4 PRINT#10,CHR$(15)"COMMODORE "
5 PRINT#10,CHR$(14)" MPS-803 ";
6 PRINT#10,CHR$(15)" MPS-803 "
7 FORK=65TO68
8 PRINT#10,CHR$(14)CHR$(K)CHR$(15)CHR$(K);
9 NEXTK
10 PRINT#10,CHR$(15):CLOSE10
```

In questo caso stampiamo con PRINT#10 e non trasferendo la stampa dal video alla stampante con CM10.
Nota alle linee 7/9 come otteniamo in ciclo la stampa alternata di un carattere allargato e uno normale; inoltre scriviamo le variabili stringa una vicino all'altra senza punteggiatura, ma dobbiamo porre un ";" finale per evitare che vada a capo.

RISULTATI PROGRAMMA COD14-2

```
COMMODORE  COMMODORE
 MPS-803   MPS-803
AABBCCDD
```

CHR$(15) MODO CARATTERE NORMALE

Predispone la stampa nel modo normale, che è attivo all'accensione. Devi usare questo codice per disattivare sia il modo a carattere allargato, che il modo grafico.

CHR$(16) SPOSTAMENTO POSIZIONE STAMPA A UNA COLONNA

Questo codice predispone l'inizio della stampa a una colonna, tra 00 e 79. Il numero della colonna, espresso come stringa di 2 caratteri, deve seguire immediatamente CHR$(16). Ricorda che le cifre numeriche hanno codice ASCII che varia da 48 a 57; per indicare la colonna 18 puoi scrivere "18" oppure CHR$(49)CHR$(56).
Il codice 16 può essere usato sia in modo testo che in modo grafico. Riportiamo un esempio nel programma COD16-1.
Dopo l'esecuzione del programma devi eseguire in modo immediato:

PRINT#10:CLOSE10.

```
0123456789012345678901234567890123456789012345678 9
 0 X                1 X                2 X
0123456789012345678901234567890123456789012345678 9
X 0                X 1                X 2
```

```
1 REM COD16-1
5 OPEN10,4
10 CMD10:A$=""
15 DATA0,65,67,102,61,31,61,102,67,65,0
20 DATA48,48,49,53,51,48
21 DATA48,48,49,53,51,48
25 FORI=1TO11:READA
30 A$=A$+CHR$(A+128)
35 NEXTI
40 GOSUB100
50 FORI=0TO2:READX,Y
55 PRINTCHR$(15)CHR$(16)CHR$(X)CHR$(Y);I;CHR$(8)A$;
60 NEXTI:PRINTCHR$(15)
65 GOSUB100:FORI=0TO2:READX,Y
70 PRINTCHR$(8)CHR$(16)CHR$(X)CHR$(Y);A$;CHR$(15);I;
75 NEXTI
80 PRINTCHR$(15):LIST
100 FORK=0TO4:FORI=0TO9:PRINTCHR$(48+I);
105 NEXTI:NEXTK:PRINT:RETURN
```

COMMENTO A COD16-1
..5/10: apre il file logico 10 per la stampante, trasferisce l'uscita video alla stampante e pulisce la stringa A$.
..15: linea DATA che contiene i codici dell'omino con le braccia alzate, precedentemente preparato; esso occupa 11 colonne di punti.
..20: linea DATA che contiene 3 coppie di cifre, in codice ASCII, per definire le tre colonne: 00, 15, 30.
..25/35: preparazione in A$ del disegno dell'omino, aggiungendo 128 a ogni codice e trasformandolo in stringa.
..40: esecuzione del sottoprogramma in 100 per stampare una linea di numerazione delle posizioni di stampa.
..50/60: stampa in ciclo, dopo aver letto le cifre di ogni colonna dalla linea DATA 20, aver definito la posizione di stampa, del numero I e dell'omino. Nota nei risultati che le colonne selezionate sono 00, 15 e 30, che i numeri sono stampati preceduti e seguiti da uno spazio, e che gli omini occupano quasi due posizioni carattere. In questo caso il codice 16 viene usato trovandosi in modo carattere per effetto del codice 15. Concluso il ciclo, va a capo e ripristina il modo carattere con il codice 15.
..65/75: esegue nuovamente il sottoprogramma per numerare le posizioni di stampa, poi passa con il codice 8 in modo grafico e usa il codice 16 per definire la posizione della colonna di inizio stampa. Nel ciclo viene stampato prima l'omino e poi il numero I, dopo essere tornati in modo carattere con il codice 15. Nota nei risultati che: gli omini iniziano esattamente alle colonne 00, 15 e 30 (la numerazione va da 0 a 79), mentre i numeri iniziano esattamente 8 punti dopo l'omino, cioè risultano sfalsati rispetto alla sovrastante colonna di numerazione.
..80: ripristino del modo carattere e lista del programma.
..100/105: sottoprogramma di numerazione linea di stampa da 0 a 9 per 5 volte.

CHR$(17) PASSAGGIO AL SET MINUSCOLO/MAIUSCOLO

Questo codice fa passare al set minuscolo/maiuscolo, con validità locale, cioè solo per l'istruzione di PRINT in corso (fino al primo RETURN), indipendentemente dal set selezionato con l'istruzione OPEN, che torna attivo al termine della PRINT. Osserva più avanti i 4 programmi esempio da COD17/145-1 a COD17/145-4; come puoi vedere, il listato del programma esce con il set selezionato dall'istruzione OPEN, indipendentemente dall'uso del codice 17 nell'ultima PRINT eseguita.

CHR$(18) e CHR$(146) MODI RVS-ON E RVS-OFF

Il codice 18 predispone la stampa in campo inverso: RVS-ON, mentre il codice 146 predispone la stampa normale: RVS-OFF. Il primo ha validità locale, cioè resta valido fino al primo carattere RETURN. Per questa ragione, se non si desidera

alternare sulla stessa linea i due modi di stampa, non è necessario usare il codice 146.
Il programma COD18/146-1, che segue, illustra quanto detto.
Dopo l'esecuzione del programma devi eseguire in modo immediato:

PRINT#4:CLOSE4.

```
CARATTERE NORMALE
[illegible]
[illegible]CAMBIO[illegible]CAMBIO
```

```
1 REM COD18/146-1
5 A$="CARATTERE NORMALE"
10 B$="CARATTERE IN CAMPO INVERSO"
15 C$="CAMBIO"
20 OPEN4,4:CMD4
25 PRINTA$
30 PRINTCHR$(18)B$CHR$(146)
35 FORK=1TO2
40 PRINTCHR$(18)C$CHR$(146)C$;
45 NEXTK:PRINT
50 LIST
```

Il modo RVS-ON può essere usato solo per la stampa in modo testo, carattere normale o allargato.

Questi due codici possono essere usati con lo stesso effetto per la stampa sul video.
Segue il programma GETPEEK per chiarire ulteriormente questo argomento.
Dopo l'esecuzione del programma devi eseguire in modo immediato:

PRINT#4:CLOSE4.

```
PIPPO DIRETTO E INVERSO
VALORI LETTI CON PEEK:
 16  9  16  16  15  144  137  144  144  143
VALORI LETTI CON GET:
 80  73  80  80  79  80  73  80  80  79
CARATTERI IN CAMPO INVERSO:
[illegible]
```

```
1 REM GETPEEK
10 PRINT"[illegible]PIPPO[illegible]PIPPO[illegible]"
15 FORK=0TO9:A(K)=PEEK(1024+K):NEXTK
20 PRINT"[illegible]";:OPEN3,3
25 FORK=0TO9:GET#3,A$(K):NEXTK
30 PRINT#3,"[illegible]";
35 FORK=0TO9:PRINT#3,A(K);:NEXTK:PRINT#3
40 FORK=0TO9:PRINT#3,ASC(A$(K));:NEXTK
45 PRINT#3:CLOSE3
```

```
47 OPEN4,4:PRINT#4,"PIPPO DIRETTO E INVERSO
49 PRINT#4,"VALORI LETTI CON PEEK: "
50 FORK=0TO9:PRINT#4,A(K);:NEXTK:PRINT#4
51 PRINT#4,"VALORI LETTI CON GET: "
55 FORK=0TO9:PRINT#4,ASC(A$(K));:NEXTK:PRINT#4
57 PRINT#4,"CARATTERI IN CAMPO INVERSO: "
58 FORK=0TO9:PRINT#4,CHR$(18)A$(K);" ";
59 NEXTK:PRINT#4
99 CMD4:LIST
```

COMMENTO A GETPEEK

..10: stampa sulla prima riga del video la parola PIPPO, prima in modo normale (campo diretto), e poi in campo inverso, usando i caratteri di controllo all'interno della stringa di stampa.

..15: legge dalla memoria video, che inizia in 1024, i 10 caratteri stampati, usando la funzione PEEK, e li memorizza in A(K).

..20: porta il cursore nella prima posizione del video, e apre il video come file.

..25: legge dal file video con GET# i 10 caratteri presenti e li memorizza in A$(K).

..30: scende con il cursore sul video di 8 righe.

..35: stampa sul video i codici dei 10 caratteri letti con PEEK; essi sono i D/CODE dei caratteri ed è riconoscibile il codice che determina il campo inverso. Troviamo infatti: 16, 9, 16, 16, 15, per PIPPO in campo diretto e: 144, 137, 144, 144, 143, per PIPPO in campo inverso, cioè ogni codice è aumentato di 128.

..40: stampa sul video i codici dei caratteri letti con GET#, usando la funzione ASC. Come puoi vedere questi codici sono tutti uguali; essi infatti sono i codici ASCII dei caratteri e non si ha differenza tra diretto e inverso.

..45: chiude il file video.

..47: apre la stampante e stampa il titolo.

..49/50: stampa i valori letti con PEEK.

..51/55: stampa i valori letti con GET#.

..57: stampa il titolo.

..58/59: stampa i codici ASCII dei caratteri dopo il codice 18, ottenendo il campo inverso.

..99: trasferisce l'uscita video alla stampante e lista se stesso.

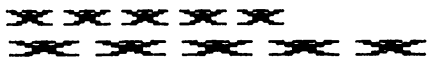
CHR$(26) RIPETIZIONE CARATTERE GRAFICO

Questo codice di controllo deve essere usato dopo essere entrati in modo grafico con il codice 8. Esso deve essere seguito da un numero N, che specifica quante volte deve essere ripetuta la colonna di punti che segue; tale numero può variare da 0 a 255 e deve essere passato con la funzione CHR$(N). Se N=0 il carattere viene ripetuto 256 volte. Se vuoi ripetere per un numero maggiore di volte devi usare la sequenza CHR$(26)CHR$(N)...; più volte. Dopo CHR$(N) deve comparire il codice che rappresenta la colonna di punti da ripetere, passato con la funzione CHR$.

Il programma COD26-1 esemplifica l'uso del codice 26 per allargare il nostro omino, già usato in qualche esempio, agendo su ogni colonna di punti che lo compone.

```
1 REM COD26-1
5 OPEN10,4
10 CMD10
15 DATA0,65,67,102,61,31,61,102,67,65,0
20 FORK=1TO5
25 FORI=1TO11
30 READA:A$=CHR$(A+128)
35 PRINTCHR$(8)CHR$(26)CHR$(2)A$;
40 NEXTI
45 RESTORE:NEXTK
50 PRINT:PRINT
55 FORK=1TO5
60 FORI=1TO11
65 READA:A$=CHR$(A+128)
70 PRINTCHR$(8)CHR$(26)CHR$(3)A$;
75 NEXTI
80 RESTORE:NEXTK
85 PRINT:PRINTCHR$(15)
90 PRINT#10:CLOSE10
```

RISULTATI COD26-1



COMMENTO A COD26-1

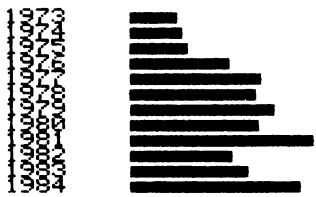
..5/10: apre la stampante con lfn=10 e trasferisce l'uscita video al file logico 10.
..15: linea DATA che contiene la codifica dell'omino, senza l'aggiunta di 128 ad ogni codice.
..20/45: ripete ciclicamente 5 volte la stampa dell'omino. Legge un codice per volta, aggiunge 128, trasforma in stringa in A$, stampa 2 volte ogni colonna di punti con la sequenza CHR$(8)CHR$(26)CHR$(2)A$. Il codice 8 poteva essere usato una sola volta fuori dal ciclo. RESTORE rende di nuovo attiva la linea DATA.
..50/75: esegue di nuovo la stampa allargata dell'omino ripetendo 3 volte ogni colonna di punti.
..85/90: ripristina il modo carattere e chiude correttamente.

Il codice 26 è molto utile per stampare istogrammi orizzontali. Nel programma COD26-2 abbiamo realizzato un esempio.

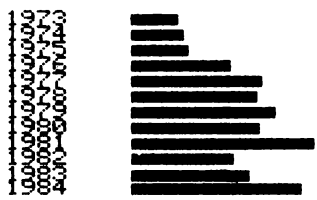
```
1 REM COD26-2
3 SA=7
5 RESTORE:OPEN10,4,SA
10 CMD10:G$=CHR$(252)
15 DATA18,20,22,38,50,48,55,49,70,39,45,65
20 PRINTCHR$(14)"ANDAMENTO VENDITE"
23 PRINT"ANNI 1973/1984"
25 PRINT
30 FORI=1TO12
35 READB
40 C=1972+I
45 PRINTCHR$(15)C;"    ";CHR$(8)CHR$(26)CHR$(B)G$
50 NEXTI
51 PRINT:PRINT
53 IFSA=7THENSA=0:PRINT#10:CLOSE10:GOTO5
55 PRINT#10,CHR$(15):CLOSE10:STOP
```

RISULTATI COD26-2

andamento vendite
anni 1973/1984



ANDAMENTO VENDITE
ANNI 1973/1984



COMMENTO A COD26-2
..3: pone SA=7, per attivare il set minuscolo/maiuscolo.
..5/10: esegue il RESTORE e apre la stampante. Trasferisce l'uscita video alla stampante e definisce il carattere grafico G$, che corrisponde a una colonna di punti con spenti i due punti più in alto; cosi le barrette dell'istogramma non si toccano.
..15: linea DATA con 12 numeri che rappresentano l'andamento delle vendite in 12 anni.
..20/25: stampa l'intestazione della tabella.
..30/51: stampa le 12 linee della tabella. Per ogni linea stampa in modo carattere l'anno e in modo grafico la barretta ripetendo il carattere G$ tante volte quanto è il valore B. Nota che va a capo in modo grafico, e quindi i numeri degli anni risultano un po' ravvicinati verticalmente.
..53: se SA=7 pone SA=0 e ritorna alla linea 5 dopo aver chiuso il file. Così stampa una seconda volta la tabella, ma con l'intestazione nell'altro set di caratteri.
..55: se SA=0 ritorna in modo carattere e chiude correttamente.

CHR$(27) SPOSTAMENTO POSIZIONE GRAFICA

Consente di spostare la posizione di inizio della stampa in uno dei 480 punti di una linea, da 0 a 479.
Esso può essere usato sia in modo testo che in modo grafico. Non può però essere usato da solo, ad esso deve seguire il codice 16, seguito a sua volta dalla posizione del punto espressa in due byte, HI e LO, passati con la funzione CHR$. Per esempio se vuoi stampare a partire dal punto 323, devi eseguire il seguente calcolo:

```
X=INT(323/256)
Y=323-X*256
```

e usare nell'istruzione PRINT la sequenza:

```
CHR$(27)CHR$(16)CHR$(X)CHR$(Y)...
```

Nel programma COD27-1 stampiamo un gruppo di omini sovrapposti iniziando la prima volta nel punto 33, la seconda nel punto 22, la terza nel punto 11 e la quarta nel punto 0. I due byte HI e LO che danno la posizione di inizio devono essere sempre passati, anche se il byte HI è nullo.

```
1 REM COD27-1
5 OPEN10,4,7:CMD10
10 DATA0,65,67,102,61,31,61,102,67,65,0
15 A$(I)="":FORJ=1TO11:READA:A=A+128
20 A$=A$+CHR$(A):NEXTJ
25 FORK=0TO4
30 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
35 PRINTCHR$(11*(3+K))A$;
40 NEXTK:PRINT
45 FORK=0TO6
50 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
55 PRINTCHR$(11*(2+K))A$;
60 NEXTK:PRINT
65 FORK=0TO8
70 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
75 PRINTCHR$(11*(1+K))A$;
80 NEXTK:PRINT
85 FORK=0TO10
90 PRINTCHR$(8)A$;
95 NEXTK:PRINT
100 PRINT:PRINTCHR$(15)
105 PRINT#10:CLOSE10
```

RISULTATI COD27-1

COMMENTO A COD27-1

..5: apre il file della stampante con lfn=10 e trasferisce l'uscita video ad esso.
..10: linea DATA che definisce l'omino.
..15/20: costruzione in A$ dell'omino.
..25/40: stampa ciclica di 5 omini a partire dalle colonne: 33, 44, 55, 66, 77.
..45/60: stampa ciclica di 7 omini a partire dalle colonne: 22, 33, 44, 55, 66, 77, 88.
..65/80: stampa ciclica di 9 omini a partire dalle colonne: 11, 22, 33, 44, 55, 66, 77, 88, 99.
..85/95: stampa ciclica di 11 omini a partire dalla colonna 0.
..100/105: ritorno alla stampa in modo carattere e chiusura corretta.
Nella stampa delle prime 3 linee bastava posizionarsi al primo punto, dopo essere passati in grafica, fuori ciclo e poi proseguire la stampa in ciclo senza ulteriori posizionamenti.

Nel programma COD27-2 stampiamo 7 tacche nelle posizioni punto 0, 50, 100, 150, 200, 250 e 300. Abbiamo stampato una linea di numeri per rendere riconoscibili le posizioni delle tacche.

```
1 REM COD27-2
3 OPEN10,4
5 CMD10
7 FORK=0TO4:FORI=0TO9:PRINTCHR$(48+I);
9 NEXTI:NEXTK:PRINT
11 FORI=0TO6
13 A=50*I
15 B=INT(A/256)
17 C=A-B*256
19 PRINTCHR$(8)CHR$(27)CHR$(16);
21 PRINTCHR$(B)CHR$(C)CHR$(255);
23 NEXTI:PRINT#10,CHR$(15):CLOSE10
```

RISULTATI COD27-2

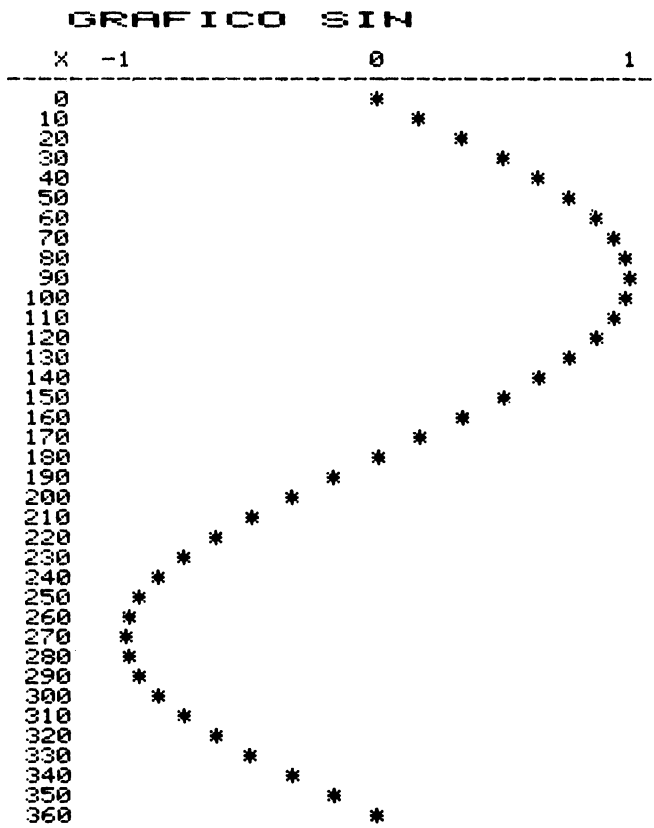
```
0123456789012345678901234567890123456789012345678 9
|         |         |         |         |         |         |
```

Nota come viene calcolata la posizione punto alle linee 13/17.

Come ultimi esempi dell'uso dei codici 27 e 16 riportiamo i programmi GRAFICO1 e GRAFICO2.

```
1 REM GRAFICO1
3 OPEN4,4:CMD4
5 D$=CHR$(14):N$=CHR$(15)
7 P$=CHR$(16):G$=CHR$(27)
9 C=23:A=16:O=4
11 A$="-":FORI=0TOC+A:A$=A$+"-":NEXT
13 S$="          "
15 PRINTD$"  GRAFICO SIN"
17 PRINTN$
19 PRINTLEFT$(S$,O-1)+"X";
21 PRINTSPC(C-A-O-1)"-1";
23 PRINTSPC(A-1)"0";
25 PRINTSPC(A-1)"1"
27 PRINTA$
29 FORI=0TO360STEP10
31 I$=RIGHT$(S$+STR$(I),O)
33 YO=C*6+A*6*SIN(I*π/180)
35 YH=INT(YO/256):YL=YO-YH*256
37 PRINTI$G$P$CHR$(YH)CHR$(YL)"*"
39 NEXTI:PRINT#4,N$:CLOSE4:STOP
```

RISULTATI GRAFICO1



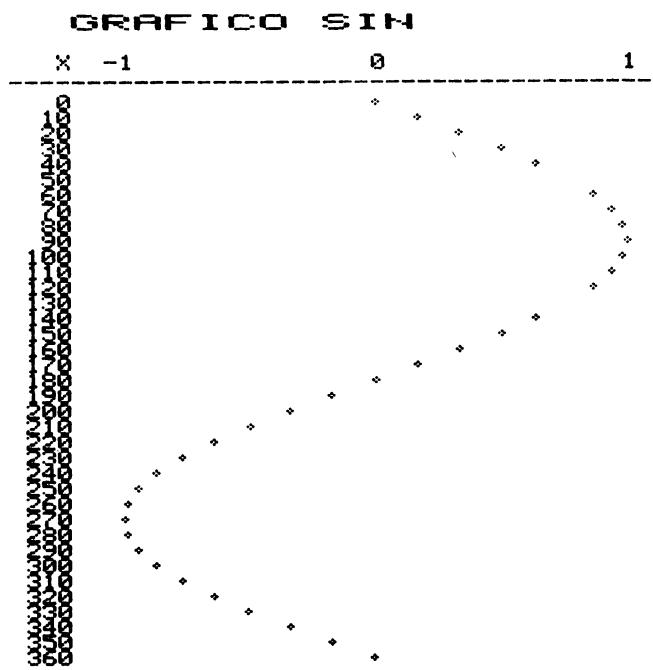
COMMENTO A GRAFICO1

..3: apre la stampante e trasferisce ad essa l'uscita video.
..5/7: definisce come stringhe i caratteri di controllo 14, 15, 16 e 27.
..9: inizializza alcune costanti.
..11/13: prepara le stringhe A$ e S$.
..15: stampa l'intestazione in caratteri allargati.
..17/25: stampa X, —1, 0, 1 usando la funzione SPC per posizionarsi.
..27: stampa le lineette.
..29/39: ciclo di stampa della funzione SIN tra 0 e 360, calcolando le coordinate dei punti e stampandoli con il carattere asterisco in modo testo. Il posizionamento avviene con CHR$(27)CHR$(16) seguiti dalla posizione punto calcolata alle linee 33/35.
Questo grafico è ottenuto lavorando in modo testo.

Il programma GRAFICO2, invece, lavora in modo grafico, stampando un piccolo rombo definito in F$ con 3 colonne di punti. Come puoi vedere dai risultati il grafico risulta molto più compatto.

```
1 REM GRAFICO2
3 OPEN4,4:CMD4
5 D$=CHR$(14):N$=CHR$(15):GR$=CHR$(8)
7 P$=CHR$(16):G$=CHR$(27):NO$=CHR$(15)
9 C=23:A=16:O=4:F$=CHR$(136)+CHR$(148)+CHR$(136)
11 A$="-":FORI=0TOC+A:A$=A$+"-":NEXT
13 S$="          "
15 PRINTD$"  GRAFICO SIN"
17 PRINTN$
19 PRINTLEFT$(S$,O-1)+"X";
21 PRINTSPC(C-A-O-1)"-1";
23 PRINTSPC(A-1)"0";
25 PRINTSPC(A-1)"1"
27 PRINTA$
29 FORI=0TO360STEP10
31 I$=RIGHT$(S$+STR$(I),O)
33 YO=C*6+A*6*SIN(I*π/180)
35 YH=INT(YO/256):YL=YO-YH*256
37 PRINTNO$I$GR$G$P$CHR$(YH)CHR$(YL)F$
39 NEXTI:PRINT#4,N$:CLOSE4:STOP
```

RISULTATI GRAFICO2



CHR$(145) PASSAGGIO SET MAIUSCOLO/GRAFICO

Questo codice fa passare al set maiuscolo/grafico in modo temporaneo, cioè con validità limitata, fino al primo carattere RETURN. Rimane preponderante la definizione del set di caratteri operata con il valore di "sa" al momento della OPEN.
Seguono 5 programmi esempio, nei quali si usa il codice 145 e il codice 17 in diverse combinazioni con il valore di SA. Ogni programma è preceduto dai suoi risultati, dato che termina listando se stesso.
Dopo l'esecuzione di ogni programa devi eseguire in immediato:

PRINT#10:CLOSE10

per chiudere correttamente

```
commodore
COMMODORE


1 rem cod17/145-1
2 open10,4,7
3 cmd10
4 printchr$(17)"commodore"
5 printchr$(145)"commodore"
6 printchr$(17):print
7 list
```

```
commodore
COMMODORE

1 rem cod17/145-2
2 open10,4,7
3 cmd10
4 printchr$(17)"commodore"
5 printchr$(145)"commodore"
6 list
```

```
commodore
COMMODORE


1 REM COD17/145-3
2 OPEN10,4
3 CMD10
4 PRINTCHR$(17)"COMMODORE"
5 PRINTCHR$(145)"COMMODORE"
6 PRINTCHR$(17):PRINT
7 LIST
```

```
commodore
COMMODORE

1 REM COD17/145-4
2 OPEN10,4
3 CMD10
4 PRINTCHR$(17)"COMMODORE"
5 PRINTCHR$(145)"COMMODORE"
6 LIST
```

```
ABCDEFGHIabcdefghi
AaBbCcDdEeFfGgHh
↑A B C D E F G H

1 REM COD17/145-5
2 S1$=CHR$(145):S2$=CHR$(17)
3 OPEN10,4:CMD10
4 PRINTS1$"ABCDEFGHI";
5 PRINTS2$"ABCDEFGHI"
6 FORK=0TO7
7 PRINTS1$CHR$(65+K)S2$CHR$(65+K);
8 NEXTK:PRINT
9 FORK=0TO7
10 PRINTS1$CHR$(97+K)S2$CHR$(97+K);
11 NEXTK:PRINT
12 LIST
```

2.4 GESTIONE DEL BUFFER DI STAMPA

Il buffer della MPS-803 può contenere 90 caratteri si ha la stampa automatica quando il buffer è pieno. La stampa può aver luogo producendo lo svuotamento totale o parziale del buffer. Nel buffer vengono memorizzati tutti i caratteri che il calcolatore invia, quelli stampabili, i caratteri separatori, le funzioni di stampa, i caratteri di controllo.
Quando usi un programma che manda dati alla stampante, non sempre vedrai uscire dati in corrispondenza all'esecuzione di istruzioni PRINT; se non operi bene puoi chiudere il file di stampa con CLOSE senza aver svuotato completamente il buffer (questo non succede se prima della CLOSE usi un'istruzione PRINT senza lista dati).
Riassumiamo le condizioni nelle quali avviene la stampa:
1) Il buffer è pieno, cioè contiene 90 caratteri, ma i caratteri di testo stampabili sono meno di 80, o tra caratteri di testo e caratteri grafici sono presenti meno di 480 punti. Il buffer viene svuotato completamente producendo una linea di stampa senza andare a capo (infatti non è stato incontrato un carattere di codice 10 o 13 che manda a capo).

2) Il buffer contiene meno di 90 caratteri, ma riceve un carattere di "vai a capo"; si ha la stampa di tutti i caratteri con svuotamento completo del buffer e si va a capo.
3) Il buffer non è pieno, ma contiene tra caratteri di testo e grafici più di 480 punti stampabili; si ha la stampa di una linea con vai a capo, e il buffer viene svuotato dei caratteri stampati.

Il programma ST-AUTOM esemplifica quanto detto.

```
1 REM ST-AUTOM
2 M0$="STAMPA 80 CARATTERI, CHR$(13) FINALE"
3 M1$="STAMPA 90 CARATTERI, CHR$(13) FINALE"
4 M2$="STAMPA 90 CARATTERI, CON ; FINALE"
5 M3$="STAMPA 8 VOLTE CHR$(10); E POI 70"
6 M3$=M3$+" CARATTERI, CON CHR$(13); FINALE"
7 LF$=CHR$(10):CR$=CHR$(13)
8 A$="0123456789":B$=""
9 A1$="ABCDEFGHIJKLMNOPQRSTUVWXYZ":B1$=""
10 FORK=1TO8:B$=B$+A$:NEXTK
11 FORK=1TO4:B1$=B1$+A1$:NEXTK
12 M4$="STAMPA 12 VOLTE CHR$(13); POI 26 "
13 M5$=LEFT$(M4$,26)
14 M4$=M4$+"CARATTERI CON CHR$(13) FINALE"
15 M5$=M5$+"POI LE 10 CIFRE CON ; FINALE"
101 OPEN4,4
103 PRINT#4,M0$
105 PRINT#4,B$:GOSUB200
107 PRINT#4,M1$
109 PRINT#4,B$;A$:GOSUB200
111 PRINT#4,M2$
113 PRINT#4,B$;A$; :GOSUB200
115 PRINT#4,M3$
117 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;
119 PRINT#4,LF$;LEFT$(B$,70);CR$;
121 PRINT#4,M4$
123 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;
125 PRINT#4,CR$;CR$;CR$;CR$;CR$;A1$:GOSUB200
127 PRINT#4,M3$
129 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;
131 PRINT#4,LF$;LEFT$(B1$,70);CR$;
133 PRINT#4,M5$
135 PRINT#4,"LE 10 CIFRE RESTANO NEL BUFFER"
137 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;
139 PRINT#4,CR$;CR$;CR$;CR$;CR$;A$; :GOSUB200
141 CLOSE4:STOP
200 FORI=1TO1000:NEXTI:RETURN
```

Non riportiamo i risultati del programma, che potrai ottenere tu eseguendolo. Esso prepara diverse stringhe di diversa lunghezza e le stampa; dopo ogni stampa viene chiamata una routine che provoca un ciclo di attesa e che ti permette di vedere cosa è stato stampato fino a quel momento. Ricorda che se la lista di stampa termina senza punteggiatura il sistema aggiunge CHR$(13) e quindi si ha la stampa con a capo. Se la lista di stampa termina con ";" non viene aggiunto alcun carattere, mentre se essa termina con "," si ha l'aggiunta di 10 spazi.

Nel programma ST-AUTOM, dato che l'ultima lista di stampa termina con "," (linea 139) e alla linea 141 si ha la CLOSE del file, gli ultimi caratteri restano nel buffer e non sono stampati. Prova alla fine del programma ad eseguire in immediato:

```
OPEN4,4:PRINT#4:CLOSE4
```

e li vedrai uscire.

2.5 COME SI COMPONE UNO STAMPATO

Nella preparazione degli stampati si presentano due problemi: gli allineamenti orizzontali e le spaziature verticali. Per quanto riguarda il primo problema si può agire sulla posizione di stampa con:
..le funzioni TAB e SPC,
..la punteggiatura "," e ";",
..i codici di controllo 16 e 27 seguiti da opportuni parametri,
..le funzioni LEN, LEFT$, MID$ e RIGHT$, che consentono di preparare dati stringa tutti della stessa lunghezza.
In generale si allineano i numeri a destra o al punto decimale, e le parole a sinistra.
Per quanto riguarda il problema delle spaziature verticali, dato che la nostra stampante non contiene un contarighe automatico (con relativi codici di controllo per usufruirne), dobbiamo creare noi una routine che conti le righe di avanzamento della carta e ci consenta di intervenire per andare a nuovo foglio o per posizionarci a una determinata riga.
È necessario conoscere il numero di righe del modulo che si usa e stabilire di quante righe deve essere il margine non stampato.
Il sottoprogramma CONTARIGHE ci consente di stampare su un foglio lungo 66 righe, con un margine di 6 righe. Se scriviamo GOSUB7000 otteniamo di contare una riga di stampa, andando a nuovo foglio se sono già state scritte 60 linee. Se scriviamo GOSUB7040 otteniamo di andare a nuovo foglio comunque.

```
7000 REM CONTARIGHE
7005 IFCR<>0THEN7020
7010 NR=66:REM LUNGHEZZA FOGLIO
7015 RM=6:REM RIGHE DI MARGINE
7020 IFCR=NR-RMTHEN7030:REM CAMBIO FOGLIO
7025 CR=CR+1:RETURN:REM +1 IN CONTA RIGHE
7030 FORR=1TORM:PRINT#4:NEXTR:REM CAMBIA FOGLIO
7035 CR=1:RETURN:REM RICOMINCIA FOGLIO
7040 REM ENTRATA PER CAMBIARE FOGLIO
7045 FORR=CR+1TOMR+NR:PRINT#4:NEXTR:GOTO7035
```

Quando si chiama la prima volta il sottoprogramma con GOSUB7000 deve essere CR=0, e allora viene inizializzato NR=66 e RM=6 (puoi cambiare queste costanti secondo le tue esigenze). Quando, invece CR<>0, la routine va a nuovo foglio se necessario e incrementa il contatore di riga. L'entrata 7040 fa andare a nuovo foglio indipendentemente dal numero di righe già stampate.

Seguono alcuni esempi relativi agli allineamenti orizzontali; essi stampano i risultati e poi listano se stessi (puoi evitare la lista cancellando LIST).
Dopo l'esecuzione di ogni programma devi eseguire in immediato:

PRINT#4:CLOSE4

per chiudere correttamente.

Il programma INC1 esemplifica l'uso della funzione TAB, dopo aver trasferito la stampa dal video alla stampante con CMD.

```
0123456789012345678901234567890123456789
POS0      POS10                POS30

NUOVA RIGA
```

```
1 REM INC1
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
11 PRINTTAB(0)"POS0";TAB(6)"POS10";
13 PRINTTAB(15)"POS30"
15 PRINTTAB(80);"NUOVA RIGA"
17 PRINT:LIST
```

Il programma INC2 esemplifica l'uso della funzione SPC, dopo aver trasferito la stampa dal video alla stampante con CMD.

```
0123456789012345678901234567890123456789
DIECI CAR.            DIECI CAR.
DIECI CAR.            DIECI CAR.
```

```
1 REM INC2
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
11 C$="DIECI CAR."
13 PRINTC$;SPC(10);C$
15 PRINTC$TAB(10)C$
17 PRINT:LIST
```

Puoi modificare INC1 e INC2 per stampare con PRINT#, invece che con CMD; ottieni gli stessi risultati.

Il programma INC5 esemplifica l'uso della punteggiatura finale nella lista di stampa. Esso stampa con CMD.

```
0123456789012345678901234567890123456789
ABC           DEF
PRIMA PAROLA            SECONDA PAROLA
ABCDEF
PRIMA PAROLASECONDA PAROLA
```

```
1 REM INC5
3 OPEN4,4:CMD4
5 A$="0123456789":B$=""
7 FORK=1TO4:B$=B$+A$:NEXTK
9 PRINTB$
11 PRINT"ABC","DEF"
13 PRINT"PRIMA PAROLA","SECONDA PAROLA"
15 PRINT"ABC";"DEF"
17 PRINT"PRIMA PAROLA";"SECONDA PAROLA"
19 PRINT:LIST
```

Puoi modificare INC5, stampando con PRINT#, invece che con CMD e ottenendo gli stessi risultati.

Il programma INC7 mostra come incolonnare i numeri riducendoli tutti della stessa lunghezza, dopo averli trasformati in stringa, con l'aggiunta di spazi a sinistra, usando la funzione RIGHT$.

```
0123456789012345678901234567890123456789
 1234     13567     45     890
 5432     9876     3456     12345

TABELLA NUMERI IN COLONNA

0123456789012345678901234567890123456789
      1234      13567        45        890
      5432       9876      3456      12345
```

```
1 REM INC7
5 DIMC(7),C$(7):SP$="              "
10 OPEN4,4:CMD4
15 A$="0123456789":B$=""
20 FORK=1TO4:B$=B$+A$:NEXTK
25 PRINTB$
30 DATA1234,13567,45,890
35 DATA5432,9876,3456,12345
40 FORK=0TO7:READC(K)
45 C$(K)=STR$(C(K)):NEXTK
50 FORJ=0TO1:FORK=0TO3:PRINTC(K+J*4);"   ";
55 NEXTK:PRINT:NEXTJ
60 T$="TABELLA NUMERI IN COLONNA"
65 PRINT:PRINTT$:PRINT
70 PRINTB$
75 FORJ=0TO1:FORK=0TO3
80 PRINTRIGHT$(SP$+C$(K+J*4),10);
85 NEXTK:PRINT:NEXTJ
90 PRINT#4:CMD4:LIST
```

Il programma INC8 mostra come incolonnare i numeri ricorrendo, dopo la trasformazione in stringa, alle funzioni SPC e LEN.

```
0123456789012345678901234567890123456789
 1234     8976     3456     13567
 45     890     5432     9876

TABELLA NUMERI IN COLONNA

0123456789012345678901234567890123456789
       1234       8976       3456      13567
         45        890       5432       9876
```

```
1 REM INC8
5 DIMC(7),C$(7):SP$="              "
10 OPEN4,4:CMD4
15 A$="0123456789":B$=""
20 FORK=1TO4:B$=B$+A$:NEXTK
25 PRINTB$
30 DATA1234,8976,3456,13567,45,890,5432,9876
35 FORK=0TO7:READC(K):C$(K)=STR$(C(K)):NEXTK
40 FORJ=0TO1:FORK=0TO3:PRINTC(K+J*4);"   ";
45 NEXTK:PRINT:NEXTJ
50 T$="TABELLA NUMERI IN COLONNA"
55 PRINT:PRINTT$:PRINT:PRINTB$
60 FORJ=0TO1:FORK=0TO3
65 PRINTSPC(10-LEN(C$(K+J*4)))C$(K+J*4);
70 NEXTK:PRINT:NEXTJ
75 PRINT#4:CMD4:LIST
```

Il programma INC9 mostra come allineare parole ricorrendo alle funzioni SPC e LEN.

```
BELLO  BENISSIMO  ALLEGRO  PIACEVOLE
RILASSANTE  SOLEGGIATO  GRADEVOLE  UTILE
DEFINITIVO  ARIOSO  STUPENDO  MAGNIFICO

BELLO          BENISSIMO     ALLEGRO       PIACEVOLE
RILASSANTE     SOLEGGIATO    GRADEVOLE     UTILE
DEFINITIVO     ARIOSO        STUPENDO      MAGNIFICO
```

```
1 REM INC9
5 DIMC$(11)
10 OPEN4,4:CMD4
15 DATABELLO,BENISSIMO,ALLEGRO
20 DATAPIACEVOLE,RILASSANTE,SOLEGGIATO
25 DATAGRADEVOLE,UTILE,DEFINITIVO
30 DATAARIOSO,STUPENDO,MAGNIFICO
35 FORK=0TO11:READC$(K):NEXTK
40 FORJ=0TO2:FORK=0TO3:PRINTC$(K+J*4);"   ";
45 NEXTK:PRINT:NEXTJ
50 N=0:FORK=0TO11
55 IFLEN(C$(K))>NTHENN=LEN(C$(K))
60 NEXTK:PRINT:PRINT
65 N=N+3:FORJ=0TO2:FORK=0TO3
70 PRINTC$(K+J*4);SPC(N-LEN(C$(K+J*4)));
75 NEXTK:PRINT:NEXTJ
80 PRINT#4:CMD4:LIST
```

2.6 COPIA DEL VIDEO SU CARTA

Il problema della copia su carta del video si pone nei seguenti termini:

— Sul video i caratteri sono ottenuti con matrici di punti 8x8, in ogni linea entrano 40 caratteri e le linee sono 25. Si hanno in tutto 1000 caratteri e 64000 punti.

— La stampante usa una matrice di punti 6x7 e quindi la rappresentazione dei caratteri non è uguale a quella visibile sul video. Ogni linea di stampa può contenere 80 caratteri e quindi di più di quanti ne servano per una linea video. Una linea di stampa può contenere fino a 480 punti.

— La stampante riceve dal calcolatore i caratteri da stampare sotto forma di codici ASCII, va a prelevare dalla sua ROM interna la rappresentazione dei caratteri e stampa. Essa ha bisogno di ricevere gli appositi codici di controllo per cambiare set di caratteri o per stampare in campo inverso.

— La stampante può ricevere dal calcolatore il codice per stampa grafica (8) e dopo accetta codici numerici che rappresentano una colonna di punti, formata da 7 punti.

Facciamo seguire 3 diversi programmi che stampano su carta il contenuto del video, accompagnandoli con qualche commento.

Segue il programma COPIAVIDEO1; quello che vedi non è un normale listato ottenuto con LIST, ma il programma copiato da se stesso dal video (dove naturalmente è stato ottenuto con LIST). Ti accorgi facilmente di questo osservando che le linee sono rigorosamente di 40 caratteri. I listati sono due; infatti il programma copia il video due volte cambiando il set di caratteri.

```
1 REM COPIAVIDEO1
2 REM COPIA SET MAIUSCOLO/GRAFICO
3 H1$=CHR$(145):GOSUB10000
4 H1$=CHR$(17):GOSUB10000:STOP
10000 REM HARD1
10001 OPEN4,4:PRINT#4
10002 H1=256*PEEK(648)-40
10003 FORH0=0TO24:H0$=H1$:H1=H1+40
10004 FORH2=H1TOH1+39:H3=PEEK(H2)
10005 IFH3>128THENH3=H3-128:H4=1:H0$=H0$
+CHR$(18)
10006 IF(H3>0)*(H3<32)THENH3=H3+64:GOTO1
0010
10007 IF(H3>31)*(H3<64)THEN10010
10008 IF(H3>63)*(H3<96)THENH3=H3+128:GOT
O10010
10009 IF(H3>95)*(H3<128)THENH3=H3+64:GOT
O10010
10010 H0$=H0$+CHR$(H3)
10011 IFH4=1THENH0$=H0$+CHR$(146):H4=0
10012 NEXTH2:PRINT#4,H0$:NEXTH0
10013 PRINT#4:CLOSE4:RETURN
READY.
RUN
```

```
1 rem copiavideo1
2 rem copia set maiuscolo/grafico
3 h1$=chr$(145):gosub10000
4 h1$=chr$(17):gosub10000:stop
10000 rem hard1
10001 open4,4:print#4
10002 h1=256*peek(648)-40
10003 forh0=0to24:h0$=h1$:h1=h1+40
10004 forh2=h1toh1+39:h3=peek(h2)
10005 ifh3>128thenh3=h3-128:h4=1:h0$=h0$
+chr$(18)
10006 if(h3>0)*(h3<32)thenh3=h3+64:goto1
0010
10007 if(h3>31)*(h3<64)then10010
10008 if(h3>63)*(h3<96)thenh3=h3+128:got
o10010
10009 if(h3>95)*(h3<128)thenh3=h3+64:got
o10010
10010 h0$=h0$+chr$(h3)
10011 ifh4=1thenh0$=h0$+chr$(146):h4=0
10012 nexth2:print#4,h0$:nexth0
10013 print#4:close4:return
ready.
run
```

Il programma è formato da 4 linee di testata che si limitano a preparare la stringa H1$ con il codice di controllo del set maiuscolo/grafico e a chiamare il sottoprogramma HARD1, e poi a modificare il codice di controllo per il set maiuscolo/minuscono e a richiamare ancora HARD1.
Il sottoprogramma HARD1 calcola l'indirizzo di inizio della mappa video alla linea 10002, poi con la funzione PEEK preleva i caratteri dal video (essi sono in D/CODE), controlla se sono maggiori di 128 per inviare il carattere di attivazione del campo inverso, poi li trasforma in codice ASCII. I caratteri sono ammucchiati in una stringa e poi stampati una linea video dopo l'altra. Essi vengono stampati nel set attivo sulla stampante e viene rispettato il campo inverso.

Segue il programma COPIAVIDEO2, che, come vedi, è molto semplice. Esso è formato da poche linee di testata e da un sottoprogramma. Alla linea 2 viene posto SA=0 e viene chiamato il sottoprogramma ottenendo la copia del video con il carattere standard. Alla linea 3 viene posto SA=7 e richiamato il sottoprogramma ottenendo la stampa nell'altro set. Il sottoprogramma HARD2 si limita ad aprire il video come file logico 3 e ad aprire la stampante; poi viene letto il video con 40 comandi GET#3 per riga formando una stringa di 40 caratteri che viene stampata. Alla linea 10030 si deve terminare con punto e virgola perché il carattere "a capo" viene già letto con GET#3. Anche questa volta i listati del programma sono ottenuti con l'uso del programma stesso.
Questo programma non rispetta i caratteri in campo inverso, infatti il codice ASCII ottenuto con GET non riporta notizie sul campo. Alla linea 10005 vedi in campo inverso il carattere compreso tra le virgolette.

```
READY.
LIST

1 REM COPIAVIDEO2
2 SA=0:GOSUB10000
3 SA=7:GOSUB10000
4 STOP
10000 REM HARD2
10005 PRINT"■";
10007 OPEN3,3:OPEN4,4,SA
10010 FORK=0TO24:A$=""
10015 FORJ=0TO39
10020 GET#3,B$:A$=A$+B$
10025 NEXTJ
10030 PRINT#4,A$;
10035 NEXTK:PRINT
10040 CLOSE3:CLOSE4
10045 RETURN
READY.
RUN
```

```
list

1 rem copiavideo2
2 sa=0:gosub10000
3 sa=7:gosub10000
4 stop
10000 rem hard2
10005 print"■";
10007 open3,3:open4,4,sa
10010 fork=0to24:a$=""
10015 forj=0to39
10020 get#3,b$:a$=a$+b$
10025 nextj
10030 print#4,a$;
10035 nextk:print
10040 close3:close4
10045 return
ready.
run
```

Infine segue il programma COPIAVIDEO3 che copia le prime 5 linee del video ricostruendo sulla stampante per colonne di punti quello che legge sul video. Si tratta di un lavoro abbastanza complicato e risulta lento programmato in BASIC, ma è interessante come algoritmo; infatti si lavora in BASIC a livello di bit.
La testata del programma è formata da quattro sequenze in ognuna delle quali si pulisce il video, si predispone il set di caratteri e il puntatore D alla mappa dei caratteri in ROM, si chiama il sottoprogramma in 100 che stampa sul video tutti i caratteri stampabili (in tutto sono 192) occupando quasi 5 linee. I set predisposti sono i due disponibili, prima in campo diretto e poi in campo inverso.
Il sottoprogramma HARD3 lavora così:

— Preleva i D/CODE dal video e li memorizza nella matrice X(4,39).
— Esegue il sottoprogramma in 20000, che, dopo aver reso disponibile la ROM dei caratteri (linee 20000 e 20001) va a prelevare per ogni carattere gli 8 byte che lo descrivono e li memorizza incolonnandoli nella matrice D(41,39). Dopo rilascia la ROM (linee 20020 e 20021) e termina.
— Esegue il sottoprogramma in 20035 che preleva dalla matrice D(41,39) a gruppi di 7 le linee, ricostruendo le colonne di punti da inviare come caratteri grafici alla stampante. Viene formata una stringa per ogni gruppo di 7 righe della matrice e viene stampata. Per calcolare i codici dei caratteri grafici si usa la matrice Z(39,7). Naturalmente la prima stringa stampata non rappresenta la prima linea del video, infatti ha lavorato su 7 righe della matrice e non su 8, ma con la seconda stampa si completano i caratteri e comincia ad essere stampata la seconda linea del video. Con 7 stampe si termina. La matrice D è stata dimensionata a 42 righe anche se ne vengono riempite solo 40 (5x8=40) per avere un multiplo di 7 e poter completare la stampa.

```
1 REM COPIAVIDEO3
2 REM COPIA SET CARATTERI 5 RIGHE
3 DIMX(4,39),D(41,39),Z(39,7)
50 H2$=CHR$(142)+CHR$(146)
55 GOSUB100:D=0:GOSUB10000
57 H2$=CHR$(142)+CHR$(18)
59 GOSUB100:D=0:GOSUB10000
70 H2$=CHR$(14)+CHR$(146)
75 GOSUB100:D=1:GOSUB10000
77 H2$=CHR$(14)+CHR$(18)
79 GOSUB100:D=1:GOSUB10000
81 PRINTCHR$(146)CHR$(142):STOP
100 PRINT"J";H2$;
105 FORK=32TO127:PRINTCHR$(K);:NEXTK
110 FORK=160TO255:PRINTCHR$(K);:NEXTK
115 PRINT:RETURN
10000 REM HARD3
10005 OPEN4,4
10010 H1=256*PEEK(648)-40
10015 FORH0=0TO4:H1=H1+40
10020 J=0:FORH2=H1TOH1+39
10021 X(H0,J)=PEEK(H2)
10060 J=J+1:NEXTH2:NEXTH0
10063 GOSUB20000
10064 GOSUB20035
10065 PRINT#4:CLOSE4:RETURN
20000 POKE56334,PEEK(56334)AND254
20001 POKE1,PEEK(1)AND251
20005 A=53248+D*2048
20007 FORM=0TO4:FORI=0TO39:Y=X(M,I)
20010 FORK=0TO7:D(M*8+K,I)=PEEK(A+Y*8+K)
20011 NEXTK:NEXTI:NEXTM
20020 POKE1,PEEK(1)OR4
20021 POKE56334,PEEK(56334)OR1:RETURN
20035 FORN=0TO41STEP7
20036 FORI=0TO39:FORL=0TO7:Z(I,L)=0
20037 NEXTL:NEXTI
20038 FORI=0TO39:FORM=0TO6:Y=D(M+N,I)
20043 FORL=0TO7
20045 IFINT(Y/2↑(7-L))=0THEN20060
20050 Z(I,L)=Z(I,L)+2↑M:Y=Y-2↑(7-L)
20060 NEXTL:NEXTM:NEXTI:PRINT#4,CHR$(8);
20065 A$="":FORI=0TO39:FORL=0TO7
20066 A$=A$+CHR$(Z(I,L)+128)
20067 IFI=19THENPRINT#4,A$;:A$=""
20070 NEXTL:NEXTI:PRINT#4,A$
20080 NEXTN:PRINT#4,CHR$(15):RETURN

READY.
```

RISULTATI DEL PROGRAMMA COPIAVIDEO3

Come vedi i caratteri ottenuti sono un po' diversi da quelli di tutti gli altri listati di questo libro, sono un po' più grassottelli.

Questo algoritmo programmato in linguaggio macchina sarebbe molto più veloce.

CAPITOLO 3

COSTRUZIONE DI PROGRAMMI

3.1 INTRODUZIONE

Lo scopo che ci siamo proposti di raggiungere con la stesura di questo libro è sempre stato quello di insegnare a scrivere programmi in Basic per il calcolatore COMMODORE 64. I numerosissimi esempi sempre riportati lo testimoniano continuamente. Tuttavia ogni esempio si riferisce ad uno o più casi specifici e sicuramente, soprattutto se sei un principiante, avrai delle difficoltà a "pensare come costruire un programma" che serva a risolvere un problema.

Nel Paragrafo 3 del Capitolo 1 del primo Volume abbiamo dato una specie di schema di quello che si deve fare per impostare lo studio di un lavoro da programmare per il calcolatore. A questo punto, ti può forse essere utile tornare su quelle pagine. Quello che ti raccomandiamo nuovamente è di non metterti mai a programmare con idee vaghe in testa direttamente sul calcolatore.
Una cosa che noi riteniamo molto utile è l'esame attento e critico di programmi già funzionanti e ben documentati. In questo Capitolo sono riportati tre esempi di costruzione di programmi, ci auguriamo tu possa trovarli di tuo interesse.

Una buona tecnica da usare per affrontare un problema è il METODO TOP DOWN, cioè dall'alto al basso. Esso consiste nel fare inizialmente uno schema molto generale del lavoro e poi successivamente sviluppare ogni parte affinando sempre di più l'analisi specifica. L'ultimo affinamento sono le istruzioni del programma. Per applicare questo metodo ci si può servire sia dei diagrammi a blocchi che della descrizione verbale.

Prima di procedere vogliamo spendere alcune frasi a proposito della DOCUMENTAZIONE, diciamo, ESTERNA dei programmi. La DOCUMENTAZIONE alla quale abbiamo fatto riferimento fino ad ora è quella INTERNA che deve essere di aiuto nello stendere il programma e che risulta ASSOLUTAMENTE NECESSARIA se, dopo un pò di tempo, si devono apportare modifiche al programma. E' incredibile come si dimentica facilmente quello a cui ci si è dedicati con tanta attenzione!

Per documentazione esterna intendiamo il manuale operativo che deve accompagnare ogni programma che venga prodotto per essere utilizzato da altre persone (e anche dagli autori). Il compilatore PETSPEED è accompagnato dal suo manuale operativo e così è per tutto il software che si compra. Tuttavia, ogni acquirente vorrebbe che il manuale contenesse una maggior quantità di informazioni. E' un problema molto spinoso e ben noto a chi lavora nel campo dell'informatica.

3.2 ALGORITMI DI ORDINAMENTO

Ci siamo proposti di confrontare tra loro tre algoritmi di ordinamento memorizzando i tempi di esecuzione.
Abbiamo deciso di creare un vettore di numeri interi positivi minori o uguali a 9999 estraendo N numeri a caso. Per poter provare il programma anche in tempi successivi lavorando con gli stessi numeri, abbiamo eseguito la prima estrazione di numero a caso con argomento negativo e le successive con argomento positivo, non importa quale.
Per lasciare generalità al programma, all'inizio viene chiesto il numero negativo da usare come argomento per la prima estrazione, e il numero N di numeri sui quali si vuole lavorare.
La fase iniziale del programma consiste nell'estrarre N numeri e memorizzarli in un vettore di N elementi. Dato che i metodi di ordinamento lavorano sullo stesso vettore che contiene i numeri, abbiamo dovuto usare due vettori, uno per ordinare e l'altro per mantenere i numeri nello stesso ordine iniziale, altrimenti non erano possibili i confronti sui tempi di esecuzione. Ogni volta, prima di chiamare il sottoprogramma di ordinamento si trasferiscono i numeri originali nel vettore dove vengono poi ordinati.
Gli ordinamenti vengono fatti in senso crescente.

Passiamo a descrivere la sequenza delle operazioni:

a) preparazione di alcune stringhe per intestazione risultati;
b) richiesta della base negativa S per l'estrazione e del numero N dei numeri da estrarre, con controllo di validità;
c) apertura della stampante e stampa intestazione risultati;
d) richiesta per decidere se stampare tutti i numeri o solo i tempi di esecuzione;
e) dimensionamento dei due vettori N% e P% per gli N numeri e definizione della funzione FNY per rendere i numeri estratti compresi tra 0 e 9999;
f) estrazione dei numeri a caso con controllo del tempo;
g) preparazione dei titoli e chiamata del sottoprogramma di stampa;
f) preparazione dei titoli, chiamata del sottoprogramma di ordinamento con metodo 1, chiamata del sottoprogramma di stampa;
h) preparazione dei titoli, chiamata del sottoprogramma di ordinamento con metodo 2, chiamata del sottoprogramma di stampa;

i) preparazione dei titoli, chiamata del sottoprogramma di ordinamento con metodo 3, chiamata del sottoprogramma di stampa;
l) stop del programma.

Il sottoprogramma di stampa opera così:

m) stampa un titolo, se non si vuole la stampa dei numeri va in o);
n) stampa la tabella dei numeri 5 per riga;
o) stampa il tempo che è stato precedentemente memorizzato leggendo il valore iniziale e finale del clock.

Il metodo 1, chiamato "ordinamento a bolle"; opera così:

a1) confronta i numeri a coppie, primo con secondo, secondo con terzo, fino all'ultima coppia, penultimo con ultimo; se dal confronto risulta che qualche coppia non è in ordine viene fatto lo scambio dei numeri e viene modificato un indicatore a ricordare che si è operato uno scambio;
a2) se alla fine del ciclo l'indicatore (azzerato a inizio ciclo) è rimasto a zero, significa che la tabella è stata trovata in ordine e si esce;
a3) se invece l'indicatore è stato modificato, si ricominciano i confronti dall'inizio, ma si tralascia un elemento in coda; infatti ad ogni ciclo l'elemento maggiore scende all'ultimo posto disponibile.

Tale metodo opera un numero di confronti variabile, che dipende dal disordine dei numeri.

Il metodo 2 chiamato, "ordinamento con dimezzamento dell'intervallo" assomiglia al precedente, ma invece di confrontare due elementi che si trovano vicini (intervallo 1), confronta due elementi che distano dapprima N/2, poi N/4, e così via fino ad arrivare alla distanza 1. Alla fine vengono percorsi un numero variabile di cicli con intervallo 1, fino al raggiunto ordine.
Anche qui il numero dei cicli è variabile e dipende dal disordine iniziale dei numeri.

Il metodo 3, chiamato "ordinamento a cicli fissi", opera un numero fisso di cicli di confronto. Se gli elementi sono N, esso percorre N-1 cicli, in ognuno dei quali confronta il primo elemento disponibile con tutti gli altri e memorizza l'indice dell'elemento che risulta minore e che diventa il nuovo termine di paragone. Alla fine del ciclo il termine di paragone è l'elemento minore ed esso viene spostato nella posizione del primo elemento disponibile mediante un doppio scambio tra i due elementi interessati. Al ciclo successivo si parte da una posizione più avanti; l'ultimo ciclo è il confronto tra due elementi.
Questo metodo fa sempre lo stesso numero di confronti anche se inizia a lavorare su una tabella ordinata.

Riportiamo i diagrammi a blocchi del programma principale, del sottoprogramma di stampa e dei tre sottoprogrammi di ordinamento.

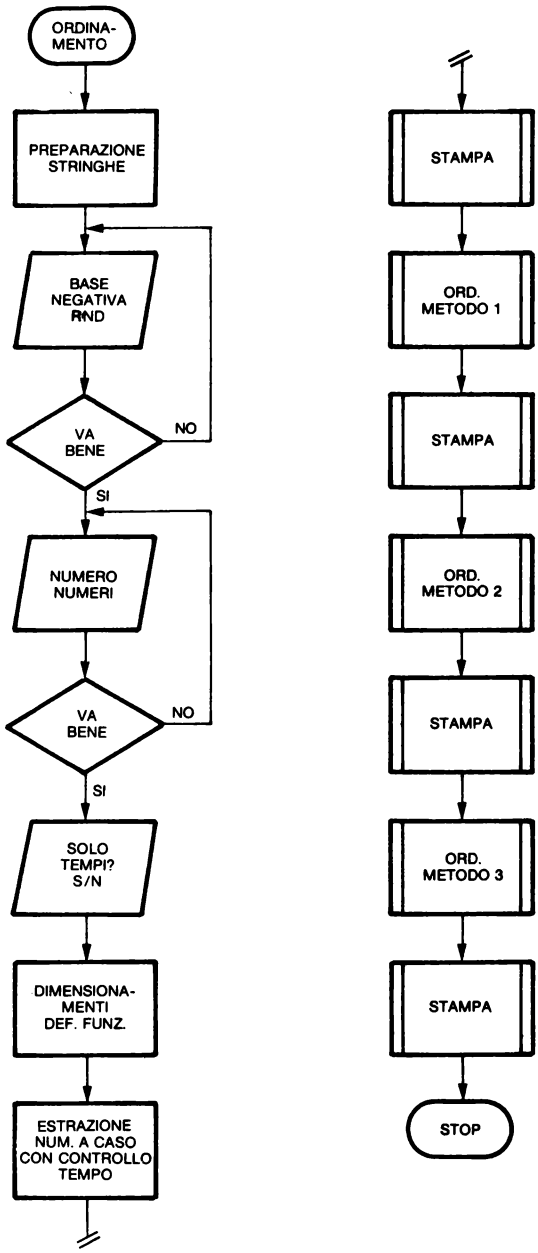


Figura 3.1 Diagramma a blocchi ORDINAMENTO

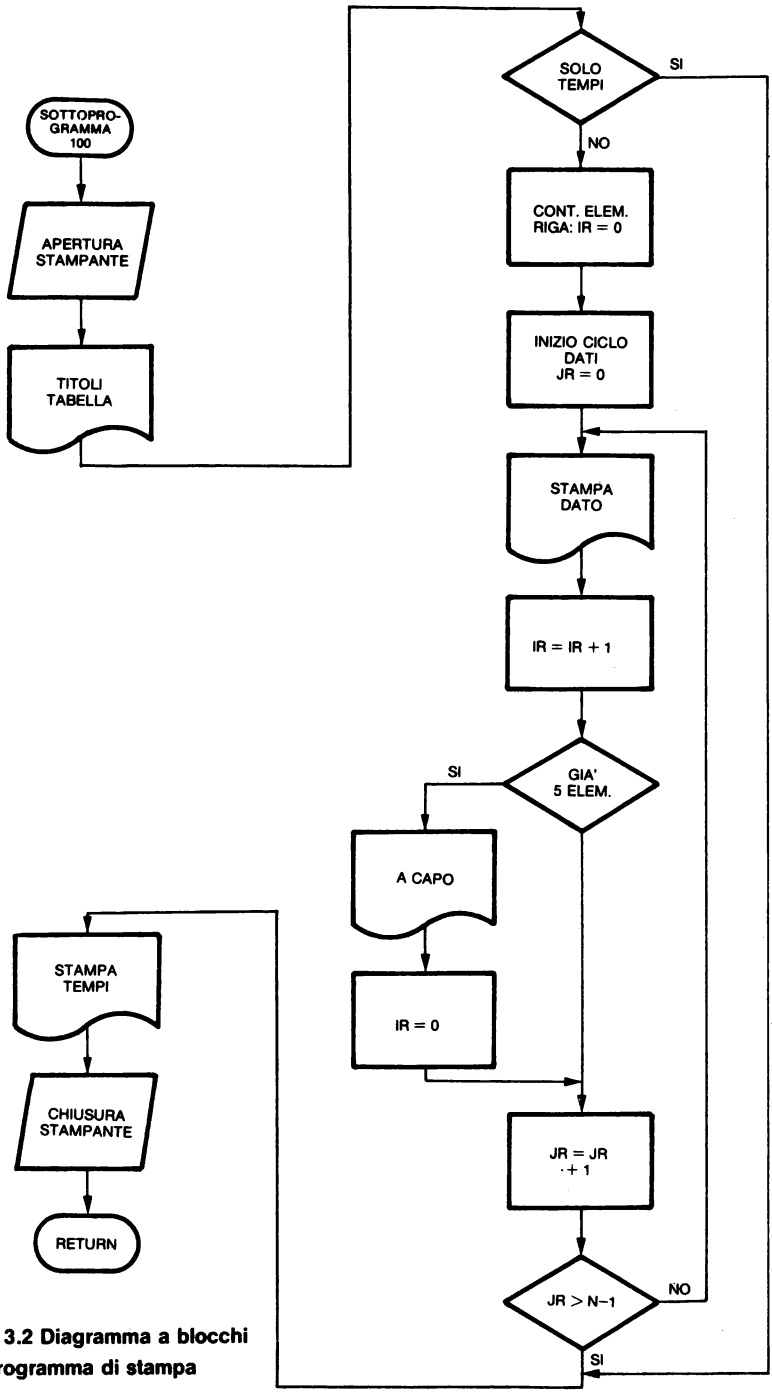


Figura 3.2 Diagramma a blocchi sottoprogramma di stampa

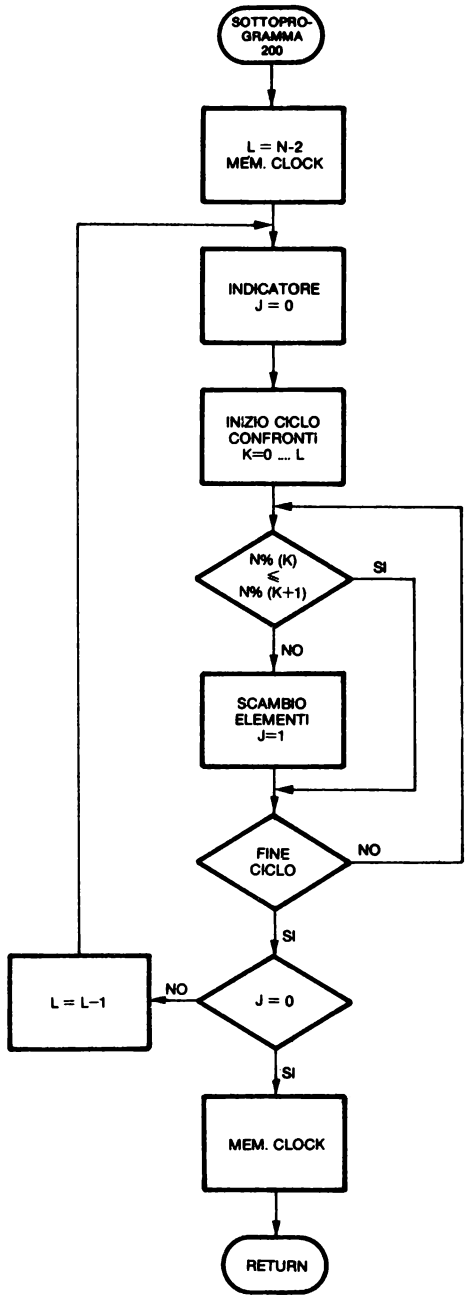


Figura 3.3 Diagramma a blocchi metodo 1 (linea 200)

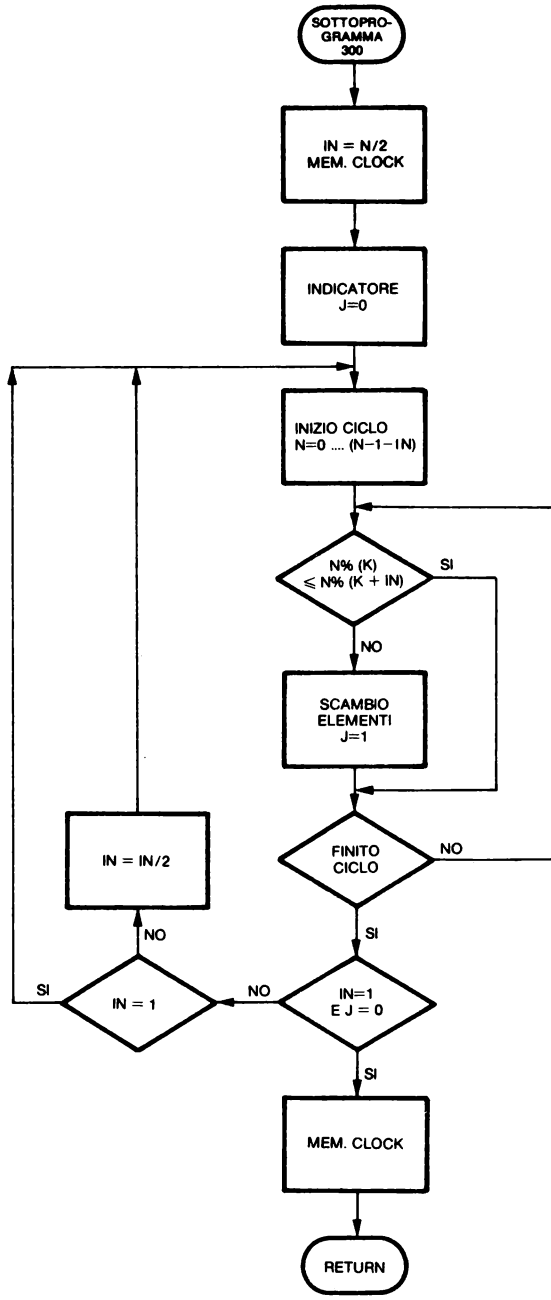


Figura 3.4 Diagramma a blocchi metodo 2 (linea 300)

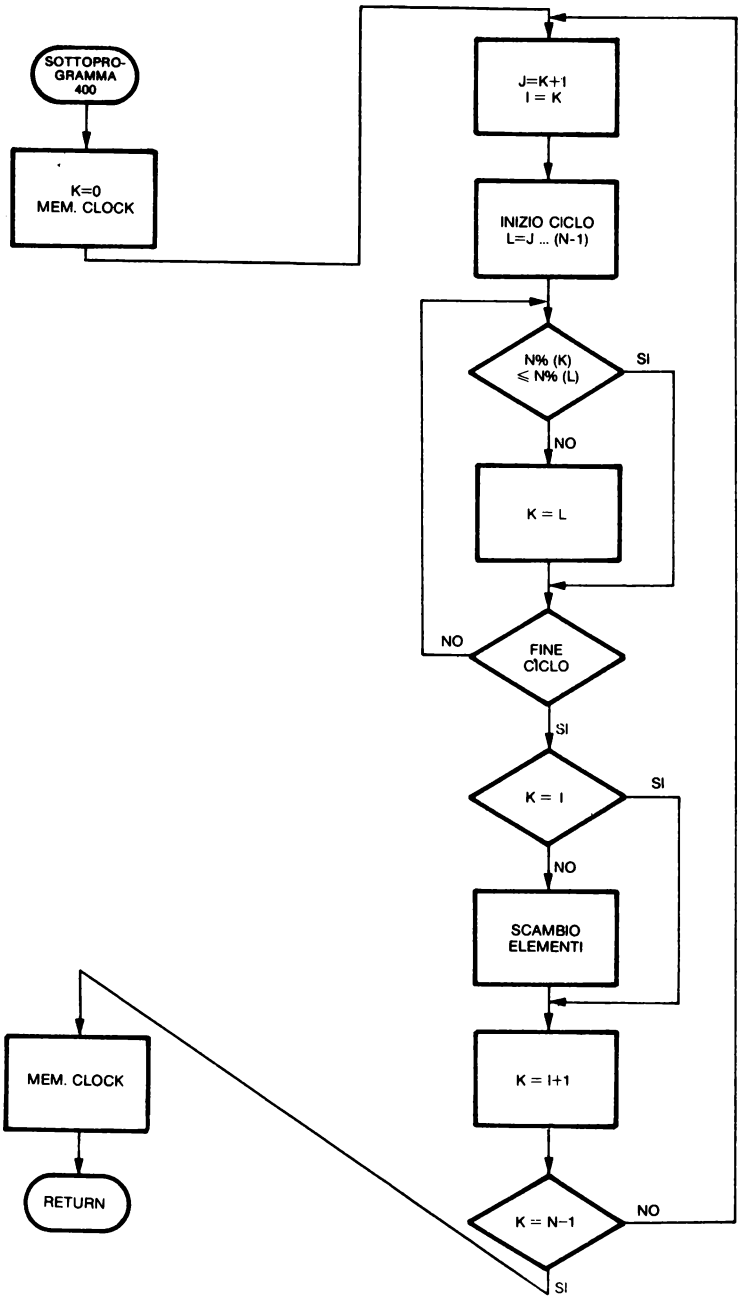


Figura 3.5 Diagramma a blocchi metodo 3 (linea 400)

Segue il listato del programma in Basic.

```
1 REM ORDINAMENTO
5 REM GENERAZIONE N NUMERI A CASO
7 REM PRIMO RND ARGOMENTO NEGATIVO
9 S$="          ":X$="ORDINATI CON METODO "
11 Y$="ORDINAMENTO"
12 INPUT"BASE NEGATIVA RND: ";S
13 IFS>=0THEN12
14 INPUT"QUANTI NUMERI: ";N
15 IFN<=0THEN14
17 OPEN4,4:CMD4:PRINT"BASE RND: ";S
19 PRINT"ORDINAMENTO DI ";N;" NUMERI":PRINT
20 PRINT#4:CLOSE4
21 REM NUMERI MINORI UGUALI A 9999
23 INPUT"STAMPA SOLO TEMPI (S/N): ";R$
25 DIM N%(N),P%(N)
30 DEFFNY(X)=X*(9999)+1
31 A=TI
35 N%(0)=FNY(RND(S)):P%(0)=N%(0)
40 FORK=1TON-1
45 N%(K)=FNY(RND(1)):P%(K)=N%(K)
50 NEXTK:B=TI
55 A$="ESTRATTI":B$="ESTRAZIONE":GOSUB100
60 C$=" 1":A$=X$+C$:B$=Y$
63 GOSUB200:GOSUB100
65 C$=" 2":A$=X$+C$:GOSUB500
67 GOSUB400:GOSUB100
69 C$=" 3":A$=X$+C$:GOSUB500
71 GOSUB300:GOSUB100
80 PRINT"BYTE LIBERI: ";FRE(0)
99 STOP
100 OPEN4,4:CMD4
105 PRINT"TABELLA NUMERI "A$:PRINT
107 IFR$="S"THEN125
110 IR=0:FORJR=0TON-1
115 N$=STR$(N%(JR)):N$=LEFT$(N$+S$,8)
117 PRINTN$;:IR=IR+1
120 IFIR=5THENPRINT:IR=0
123 NEXTJR:PRINT
125 PRINT"TEMPO "B$;(B-A)/60;" SEC."
130 PRINT#4:CLOSE4:RETURN
```

```
200 REM METODO A BOLLE
205 L=N-2:A=TI
210 J=0:FORK=0TOL
215 IFN%(K)<=N%(K+1)THEN225
220 C%=N%(K):N%(K)=N%(K+1):N%(K+1)=C%:J=1
225 NEXTK
230 IFJ=0THENB=TI:RETURN
235 L=L-1:GOTO210
300 REM METODO DIMEZZAMENTO INTERVALLO
305 IN=INT(N/2+0.5):A=TI
310 J=0:FORK=0TO(N-1-IN)
315 IFN%(K)<=N%(K+IN)THEN320
317 C%=N%(K):N%(K)=N%(K+IN):N%(K+IN)=C%:J=1
320 NEXTK
325 IFIN=1ANDJ=0THENB=TI:RETURN
327 IFIN=1THEN310
330 IN=INT(IN/2+0.5):GOTO310
400 REM METODO CICLI FISSI
405 K=0:A=TI
410 J=K+1:I=K
415 FORL=JTON-1
420 IFN%(K)<=N%(L)THEN430
425 K=L
430 NEXTL
435 IFK=ITHEN445
440 C%=N%(I):N%(I)=N%(K):N%(K)=C%
445 K=I+1:IFK=N-1THENB=TI:RETURN
450 GOTO410
500 FORK=0TON-1:N%(K)=P%(K):NEXTK:RETURN
```

VARIABILI USATE NEL PROGRAMMA

S$	stringa di 8 spazi
X$	stringa descrittiva
Y$	" "
R$	stringa per risposta s/n
A$	stringa descrittiva
B$	" "
C$	" "
N$	stringa per conversione numeri da allineare
S	base negativa estrazione numeri a caso
N	numero dei numeri su cui lavorare

N%(N-1)	vettore di numeri interi per i numeri estratti e da ordinare
P%(N-1)	vettore di numeri interi per conservare i numeri estratti
X	variabile di lavoro
A	variabile per tempo iniziale di una operazione
B	variabile per tempo finale di una operazione
K	variabile di controllo ciclo
IR	variabile per contare i numeri stampati su una riga
JR	variabile controllo ciclo
L	puntatore per ordinamento
J	indicatore per ordinamento
C%	variabile di comodo per operare doppio scambio
IN	variabile di lavoro, intervallo confronto

Riportiamo i risultati di tre prove; per i 57 numeri riportiamo anche i listati dei numeri, per le altre due solo i tempi di esecuzione.

```
BASE RND: -5890
ORDINAMENTO DI  57  NUMERI


TABELLA NUMERI ESTRATTI

 1        9386     2926     697      6505
 2284     2676     7538     5450     4055
 1595     5109     4185     5644     8507
 5934     3863     691      8831     1688
 6656     8902     2587     4152     2840
 9343     7        6788     9839     1303
 4980     9353     8985     855      1793
 4750     1706     5087     4682     1796
 8806     5604     7490     531      6088
 7991     448      2212     660      4223
 1776     2133     5284     1088     3784
 4158     3662
TEMPO ESTRAZIONE 1.73333333  SEC.
```

TABELLA NUMERI ORDINATI CON METODO 1

1	7	448	531	660
691	697	855	1088	1303
1595	1688	1706	1776	1793
1796	2133	2212	2284	2587
2676	2840	2926	3662	3784
3863	4055	4152	4158	4185
4223	4682	4750	4980	5087
5109	5284	5450	5604	5644
5934	6088	6505	6656	6788
7490	7538	7991	8507	8806
8831	8902	8985	9343	9353
9386	9839			

TEMPO ORDINAMENTO 37.5666667 SEC.

TABELLA NUMERI ORDINATI CON METODO 2

1	7	448	531	660
691	697	855	1088	1303
1595	1688	1706	1776	1793
1796	2133	2212	2284	2587
2676	2840	2926	3662	3784
3863	4055	4152	4158	4185
4223	4682	4750	4980	5087
5109	5284	5450	5604	5644
5934	6088	6505	6656	6788
7490	7538	7991	8507	8806
8831	8902	8985	9343	9353
9386	9839			

TEMPO ORDINAMENTO 20.0333333 SEC.

TABELLA NUMERI ORDINATI CON METODO 3

1	7	448	531	660
691	697	855	1088	1303
1595	1688	1706	1776	1793
1796	2133	2212	2284	2587
2676	2840	2926	3662	3784
3863	4055	4152	4158	4185
4223	4682	4750	4980	5087
5109	5284	5450	5604	5644
5934	6088	6505	6656	6788
7490	7538	7991	8507	8806
8831	8902	8985	9343	9353
9386	9839			

TEMPO ORDINAMENTO 17 SEC.

BASE RND: -34
ORDINAMENTO DI 250 NUMERI

TABELLA NUMERI ESTRATTI

TEMPO ESTRAZIONE 7.08333334 SEC.

TABELLA NUMERI ORDINATI CON METODO 1

TEMPO ORDINAMENTO 698.8 SEC.

TABELLA NUMERI ORDINATI CON METODO 2

TEMPO ORDINAMENTO 339.166667 SEC.

TABELLA NUMERI ORDINATI CON METODO 3

TEMPO ORDINAMENTO 271.166667 SEC.

```
BASE RND: -3
ORDINAMENTO DI  503  NUMERI


TABELLA NUMERI ESTRATTI

TEMPO ESTRAZIONE 14.2  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 2904.51667  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 1370.38333  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 1593.51667  SEC.
```

Come puoi vedere dai risultati sembra in generale più veloce il metodo 3 e più lento il metodo 1. E' evidente che se si opera con il metodo 1 su una serie di numeri già in ordine si ottiene il tempo minore.

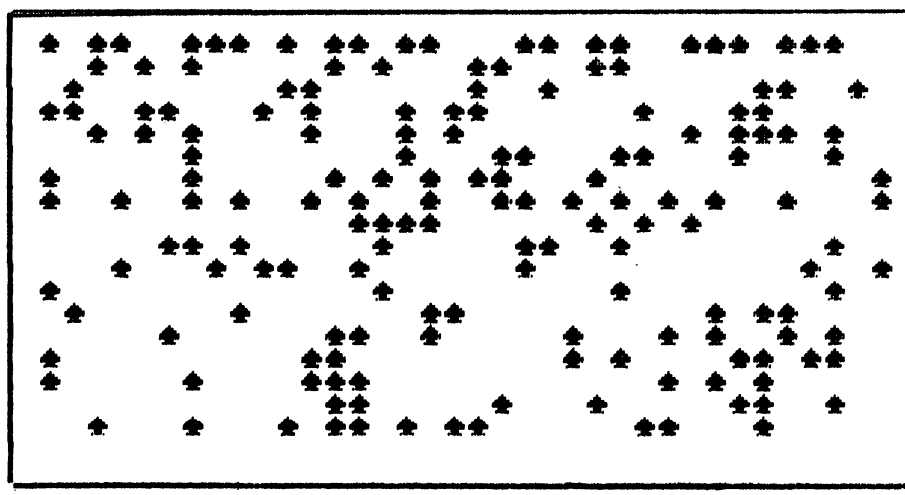
3.3 CARATTERI A CASO E COPIA SU CARTA

Abbiamo pensato di completare il programma COPIAVIDEO3 presentato nel Capitolo 2, portandolo a ricopiare su carta tutto il video e non solo 5 righe.
Per ottenere rapidamente qualcosa da ricopiare abbiamo impostato un programma che traccia una riquadratura sul video e, all'interno di questa, posiziona a caso un carattere scelto dall'utente. All'inizio viene chiesto quante volte si vuole posizionare a caso il carattere e quale carattere si sceglie. Questo non è un programma di grafica ad alta risoluzione, infatti l'argomento verrà trattato nell'apposito volume. Il video viene riempito usando i caratteri normali del sistema e non quelli che può creare l'utente.
Il programma TUTTOVIDEO con HARD5, resta però valido anche se il video è riempito con grafica ad alta risoluzione, infatti esso lavora per punti e non per

caratteri, anzi per colonne di punti (7 per volta).

Per far girare il sottoprogramma per tutto il video abbiamo dovuto fare un pò di conti: il video contiene in tutto 1000 caratteri e nella MAPPA VIDEO si trovano 1000 codici che fanno da puntatori alla descrizione dei caratteri (sia essa in ROM o in RAM). Per poter sviluppare la descrizione di ogni carattere in 8 byte occorrono 1000x8=8000 byte. Il Basic non ci consente di indirizzare i singoli byte, però se usiamo variabili con indice intere, esso usa 2 byte per ogni elemento, quindi per i nostri 8000 byte descrittori ne impegnamo 16000 sotto forma di variabili intere con indice. Con artifici di programmazione si potrebbe mettere in ogni variabile intera con indice 2 byte descrittori, infatti ognuno di essi è minore di 255, ma non vogliamo complicarci troppo la vita.
Risulta chiaro che un buon programma di HARD COPY deve essere scritto in linguaggio macchina. Noi con la scusa della copia del video riusciamo ad impostare discorsi interessanti ai fini dell'apprendimento della programmazione in Basic e questo è il nostro scopo attuale.
A questo punto non è stato difficile modificare il programma COPIAVIDEO3. Abbiamo dimensionato le 3 matrici di variabili intere:
.. X%(24,39) per contenere la MAPPA VIDEO;
.. D%(202,39) per contenere la descrizione in 8 byte di ogni carattere del video, 25x8=200, ma abbiamo dovuto aggiungere 3 per avere un multiplo di 7;
.. Z(39,7) per sviluppare le colonne di punti, 8 per ogni posizione video.
Alla linea 10015 il ciclo va da 0 a 24; alla linea 20007 il ciclo va da 0 a 24; alla linea 20035 il ciclo va da 0 a 202.Per il resto nessuna modifica.

Segue un risultato del programma:



Segue il listato del programma DISE1.

```
1 REM DISE1
2 PRINT"QUANTE ESTRAZIONI: ";:INPUTZ
3 IFZ<0THEN2
5 V$="                                    "
7 PRINT"QUALE CARATTERE: ";:INPUTA$
9 IFLEN(A$)<>1THEN7
10 PRINT"";:FORL=1TO38:PRINT"";:NEXTL:PRINT
11 PRINTLEFT$(V$,23)"";
13 FORL=1TO38:PRINT"";:NEXTL
15 PRINT"":FORL=1TO21:PRINT" ":NEXTL
17 PRINT"":FORL=1TO21:PRINTTAB(38)" ":NEXTL
20 FORK=1TOZ
23 X1=RND(3):Y1=RND(3)
27 X1=INT(X1*37+1):IFX1<2THENX1=2
29 Y1=INT(Y1*20+1):IFY1<3THENY1=3
31 PRINTLEFT$(V$,Y1);TAB(X1);A$:PRINT""
37 NEXTK
500 REM TUTTOVIDEO3
505 DIMX%(24,39),D%(202,39),Z%(39,7)
510 H1$=CHR$(145):H2$=CHR$(142)+CHR$(146)
515 D=0:GOSUB10000
525 STOP
10000 REM HARD5
10005 OPEN4,4:PRINT#4,H1$;
10010 H1=256*PEEK(648)-40
10015 FORH0=0TO24:H1=H1+40
10020 J=0:FORH2=H1TOH1+39
10021 X%(H0,J)=PEEK(H2)
10060 J=J+1:NEXTH2:NEXTH0
10063 GOSUB20000
10064 GOSUB20035
10065 PRINT#4:CLOSE4:RETURN
20000 POKE56334,PEEK(56334)AND254
20001 POKE1,PEEK(1)AND251
20005 A=53248+D*2048
20007 FORM=0TO24:FORI=0TO39:Y%=X%(M,I)
20010 FORK=0TO7:D%(M*8+K,I)=PEEK(A+Y%*8+K)
20011 NEXTK:NEXTI:NEXTM
20020 POKE1,PEEK(1)OR4
20021 POKE56334,PEEK(56334)OR1:RETURN
```

```
20035 FORN=0TO202STEP7
20036 FORI=0TO39:FORL=0TO7:Z%(I,L)=0
20037 NEXTL:NEXTI
20038 FORI=0TO39:FORM=0TO6:Y%=D%(M+N,I)
20043 FORL=0TO7
20045 IFINT(Y%/2↑(7-L))=0THEN20060
20050 Z%(I,L)=Z%(I,L)+2↑M:Y%=Y%-2↑(7-L)
20060 NEXTL:NEXTM:NEXTI:PRINT#4,CHR$(8);
20065 A$="":FORI=0TO39:FORL=0TO7
20066 A$=A$+CHR$(Z%(I,L)+128)
20067 IFI=19THENPRINT#4,A$;:A$=""
20070 NEXTL:NEXTI:PRINT#4,A$
20080 NEXTN:PRINT#4,CHR$(15)H2$:RETURN
```

Per ottenere questo programma abbiamo lavorato così:
.. abbiamo scritto ex novo le linee da 1 a 37;
.. abbiamo verificato che il quadro video uscisse bene e abbiamo memorizzato il programma su disco;
.. abbiamo richiamato in memoria il vecchio programma COPIAVIDEO3, l'abbiamo modificato mettendo in pratica quanto detto sopra e l'abbiamo memorizzato su disco con un nome provvisorio;
.. abbiamo scritto NEW e poi abbiamo caricato in memoria il programma del quadro video già provato e corretto;
.. abbiamo letto con PEEK il contenuto dei due byte 45 e 46 e calcolato l'indirizzo di inizio delle variabili; questo indirizzo - 2 dà l'indirizzo del LINK finale del programma, quello con due zeri;
.. abbiamo scritto con POKE nei 2 byte 43 e 44 spostando l'indirizzo di inizio del programma Basic a quello calcolato dell'ultimo LINK;
.. abbiamo operato il LOAD del programma di copia video precedentemente memorizzato;
.. abbiamo scritto con POKE nei 2 byte 43 e 44 l'indirizzo solito di inizio Basic;
.. abbiamo listato il programma, l'abbiamo memorizzato su disco e DOPO provato.

Nota il DOPO; ti raccomandiamo di memorizzare sempre i programmi prima di provarli, non si sa mai quello che può succedere!

Commento al programma per linee:

.. da 1 a 2 richiesta del numero di estrazioni Z, se Z negativo viene rifatta la richiesta;
.. in 5 definizione della stringa V$; essa inizia con il carattere HOME e continua con 24 caratteri CRSR/giù, serve per muoversi in verticale sul video, usandola come

argomento della funzione LEFT$ e prendendo la parte che serve;
.. da 7 a 9 richiesta del carattere, con controllo che sia uno solo;
.. in 10 traccia il bordo superiore;
.. da 11 a 13 traccia il bordo inferiore;
.. in 15 traccia il bordo sinistro;
.. in 17 traccia il bordo destro;
.. da 20 a 37 ciclo di estrazione di Z coppie di numeri a caso con argomento di RND positivo (sequenze sempre diverse); aggiustamento di X1, coordinata orizzontale per farla muovere entro i bordi laterali; aggiustamento di Y1, coordinata verticale, per farla muovere all'interno dei bordi orizzontali; disegno del carattere nella posizione definita;
.. in 505 dimensionamento delle matrici per il sottoprogramma HARD5;
.. in 510 preparazione delle stringhe con i caratteri di controllo per la stampante e il calcolatore;
.. in 515 pone D=0 per accedere al primo set di caratteri nella ROM del calcolatore, lancia il sottoprogramma HARD5;
.. in 525 fine del programma;
.. in 10005 apre la stampante come file 4 e predispone il set normale, anche se non servirebbe, dato che si stampa in grafica;
.. in 10010 prepara H1 puntatore alla mappa video (di norma in 1024);
.. da 10015 a 10060 trasferimento della mappa video nella matrice X%; si poteva anche lavorare prelevando ogni volta dalla mappa video, ma è più comodo lavorare con le variabili con indice;
.. in 10063 chiamata del sottoprogramma 20000 che va a leggere in ROM le descrizioni dei caratteri;
.. in 10064 chiamata del sottoprogramma 20035 che stampa il video;
.. in 10065 chiusura della stampante e ritorno al programma principale;
.. da 20000 a 20001 disabilitazione I/O e interrupt per rendere disponibile la ROM dei caratteri;
.. in 20005 calcolo in base al valore di D dell'indirizzo della mappa caratteri in ROM;
.. da 20007 a 20011 prelevamento della descrizione dei caratteri e memorizzazione nella matrice D%;
.. da 20020 a 20021 disabilitazione ROM caratteri e RETURN;
.. in 20035 istituzione del ciclo FOR che gestisce le righe della matrice D% a gruppi di 7;
.. da 20036 a 20037 azzeramento della matrice Z% dove si fanno le somme dei codici per le 7 righe di punti di ogni colonna di carattere grafico;
.. in 20038 istituzione del ciclo di riga e del ciclo di colonna per la striscia di 7 linee di punti e trasferimento del byte da calcolare;
.. in 20043 istituzione del ciclo che spezza il byte in bit e gli attribuisce i giusti pesi per costruire i caratteri grafici;
.. da 20045 a 20050 calcolo;

.. in 20060 chiusura dei cicli aperti e stampa codice controllo per la grafica;
.. da 20065 a 20070 stampa dei caratteri grafici contenuti nella matrice Z% che si riferisce a una riga video; i caratteri sono 40x8=320 e quindi non possono essere ammucchiati tutti in una stringa, da cui l'analisi sul valore 19 di I per stampare il primo pezzo, poi annullare A$ e continuare;
.. in 20080 chiusura del ciclo iniziato in 20035, stampa dei codici di controllo per tornare a stampa normale e RETURN.

Il difetto di questo programma DISE1 è la lentezza; impiega circa un'ora per fare la copia del video. Abbiamo provato a compilare il programma, dopo aver apportato una modifica alla linea 10010 per togliere il riferimento ai puntatori in pagina zero (abbiamo posto l'indirizzo d'inizio della mappa video come costante). Il programma origine del compilato è memorizzato come DISE2 sul dischetto dei programmi del libro e quello compilato come DISE2.WOW. Il programma compilato impiega circa 10 minuti per fare la copia del video.

3.4 STAMPA DI UNA FATTURA PROFESSIONALE

Ci siamo proposti di scrivere un programma che consenta di stampare una fattura professionale. Una cosa molto semplice senza registrazione sui libri contabili; per quest'ultimo tipo di lavoro ti consigliamo di imparare a usare programmi della famiglia VISICALC, ne esistono varie versioni anche per il COMMODORE 64 e sono molto comodi.

Il programma deve produrre i seguenti risultati:
.. stampare il numero di copie della fattura richiesto;
.. intestare il foglio di carta con: titolo, nome e cognome, di chi emette la fattura, scrivendo in centro del foglio, in alto, a caratteri allargati;
.. porre in basso su due righe l'indirizzo del mittente e il telefono, sempre centrando la scritta e ingrandendo i caratteri;
.. porre a sinistra in alto: titolo, ragione sociale o nome e indirizzo del destinatario;
.. porre a destra il numero della fattura, la città e la data;
.. scrivere su al massimo 10 righe la causale della fattura;
.. scrivere l'importo della fattura;
.. calcolare e scrivere l'importo dell'IVA;
.. calcolare il totale e scriverlo;
.. riportare i dati anagrafici richiesti.

Per semplificare il problema abbiamo deciso che la fattura si riferisce ad un solo importo, e inoltre la causale della fattura deve essere fornita con le righe già composte per la stampa.

I dati di Input sono:
.. Il numero di copie che vuoi stampare, memorizzato in NF.
.. DATI MITTENTE: Titolo, Nome, Cognome, CAP, Città, Indirizzo, Telefono, Codice Fiscale, Partita IVA, Luogo e data di nascita. Per questi 10 dati abbiamo dimensionato il vettore M$(10) e per le relative descrizioni, da usare nella richiesta dei dati, il vettore C$(10).
.. DATI FATTURA: Numero fattura, Luogo emissione, Data fattura, per i quali abbiamo usato le variabili singole: CF$, DF$ e N$.
.. DATI DESTINATARIO: Titolo, Ragione Sociale, Indirizzo, CAP/Città. Per questi dati abbiamo dimensionato il vettore I$(4) e per le relative descrizioni il vettore CI$(4).
.. Descrizione causale fattura: per queste linee che possono essere 10 al massimo abbiamo usate il vettore Z$(10), con dimensionamento implicito.
.. Importo fattura, memorizzato in M, come numero e in M$ come stringa da stampare.
.. Percentuale IVA, memorizzata in IV.

I dati di Output sono tutti quelli ricevuti come Input, più il calcolo dell'importo IVA, memorizzato in IV$, e il totale memorizzato in FT$.

Il programma è composto da due parti:
.. ricezione dati di Input;
.. stampa ciclica del numero di copie richieste.

Avendo deciso di ingrandire i caratteri dell'intestazione del foglio, abbiamo dovuto controllare che il numero dei caratteri non superi 40. In caso si hanno 3 possibili STOP:
.. linea 194 se Titolo+Nome+Cognome (M$(1), M$(2), M$(3));
.. linea 344 per l'indirizzo (M$(4), M$(5), M$(6));
.. linea 355 per il telefono (M$(7));
e si può intervenire in immediato per accorciare le relative stringhe e poi continuare con CONT. Osserva come otteniamo la centratura con l'aggiunta del giusto numero di spazi a sinistra.

Segue uno schema o blocchi non molto dettagliato del programma.

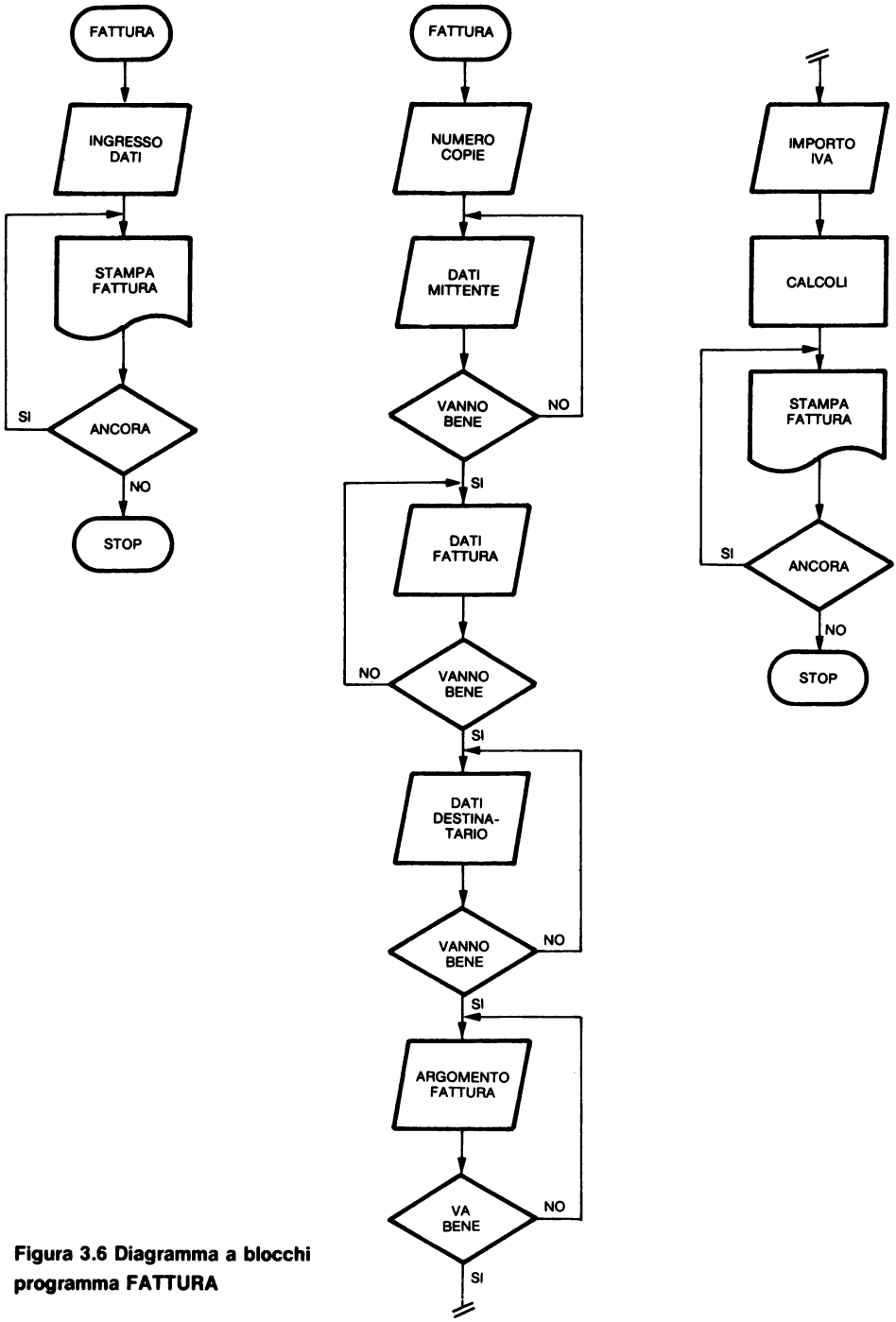


Figura 3.6 Diagramma a blocchi programma FATTURA

Segue il listato del programma FATTURA.

```
1 REM FATTURA
5 DIMM$(10),I$(4),C$(10),CI$(4)
6 C$(1)="TITOLO: ":C$(2)="NOME: "
7 C$(3)="COGNOME: "
8 C$(4)="CAP: ":C$(5)="CITTA': "
9 C$(6)="INDIRIZZO: ":C$(7)="TEL.: "
10 C$(8)="COD. FISCALE: "
11 C$(10)="LUOGO/DATA NASCITA: "
12 S3$="--------------------------":L$="L. "
13 C$(9)="PARTITA IVA: "
14 CI$(3)=C$(6):CI$(4)="CAP/CITTA': "
15 S$="":FORK=1TO40:S$=S$+" ":NEXTK
16 INPUT"▯▯▯QUANTE COPIE: ";NF
17 IFNF<0THEN16
19 V$="▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯"
20 S1$="▯▯▯DATI MITTENTE▯▯"
23 CI$(1)=C$(1):CI$(2)="RAG. SOCIALE: "
25 PRINTS1$:FORK=1TO10
30 GOSUB600
31 PRINTK$"-"C$(K);:INPUTM$(K)
33 NEXTK
35 GOSUB1000
40 IFR$="S"THEN90
45 INPUT"QUALE CAMPO: ";N
50 IFN<1ORN>10THENPRINT"▯":GOTO45
55 PRINTS1$:IFN=1THEN63
56 FORK=1TON-1
57 GOSUB600
60 PRINTK$" "C$(K)M$(K):NEXTK
63 GOSUB600
65 PRINTK$" "C$(N);:INPUTM$(N)
70 IFN=10THEN35
75 FORK=N+1TO10
76 GOSUB600
77 PRINTK$"-"C$(K)M$(K):NEXTK
80 GOTO35
90 PRINT"▯▯▯DATI FATTURA▯▯"
93 INPUT"CITTA' FATTURA: ";CF$
95 INPUT"DATA: ";DF$
97 INPUT"NUM. FATT.: ";N$
```

```
99 GOSUB1000:IFR$="S"THEN103
100 GOTO90
103 S2$="▯▯▯DATI DESTINATARIO▯▯"
105 PRINTS2$
107 FORK=1TO4
109 PRINTK;CI$(K);:INPUTI$(K)
111 NEXTK
113 GOSUB1000
115 IFR$="S"THEN133
117 INPUT"QUALE CAMPO: ";N
119 IFN<1ORN>4THENPRINT"▯▯":GOTO117
121 PRINTS2$:IFN=1THEN125
122 FORK=1TON-1
123 PRINTK;CI$(K)I$(K):NEXTK
125 PRINTN;CI$(N);:INPUTI$(N)
127 IFN=4THEN113
129 FORK=N+1TO4:PRINTK;CI$(K)I$(K):NEXTK
131 GOTO113
133 PRINT"▯▯▯ARGOMENTO FATTURA▯▯"
135 INPUT"▯QUANTE LINEE: ";ZF
137 IFZF>10THENPTRINT"▯▯":GOTO135
139 PRINT"▯";:FORK=1TOZF
141 PRINTK;": ";:INPUTZ$(K)
143 IFLEN(Z$(K))>50THENZ$(K)=LEFT$(Z$(K),50)
147 NEXTK
149 PRINT"▯";:FORK=1TOZF
151 PRINTK;"-";Z$(K):NEXTK
153 GOSUB1000
155 IFR$="S"THEN165
157 INPUT"QUALE CAMPO: ";N
159 IFN<1ORN>ZFTHENPRINT"▯▯":GOTO157
161 PRINT"RISCRIVI RIGA":INPUTZ$(N)
162 IFLEN(Z$(N))>50THENZ$(N)=LEFT$(Z$(N),50)
163 GOTO149
165 INPUT"IMPORTO: ";M:ZX=M
167 GOSUB500:M$=A$
169 INPUT"▯▯% IVA: ";IV
171 ZX=INT(M*IV/100):GOSUB500
173 IV$=A$
175 X=M+ZX:ZX=X:GOSUB500:L=LEN(A$)
177 TF$=RIGHT$(S$+A$,L)
178 M$=RIGHT$(S$+M$,L):IV$=RIGHT$(S$+IV$,L)
```

```
179 PRINT"□□POSIZIONA IL FOGLIO DI CARTA"
181 PRINT"PREMI UN TASTO PER CONTINUARE"
183 GETR$:IFR$=""THEN183
190 OPEN4,4
191 FORKF=1TONF
192 A$=M$(1)+" "+M$(2)+" "+M$(3)
194 X=LEN(A$):IFX>40THENSTOP
198 X=(80-X*2)/4
201 IFX=0THEN206
203 A$=LEFT$(S$,X)+A$
206 PRINT#4,CHR$(14)A$CHR$(15)
208 I=7:GOSUB1100
210 FORK=1TO4:PRINT#4,I$(K):NEXTK
220 I=2:GOSUB1100
225 PRINT#4,TAB(50)"NUMERO FATTURA:"N$
230 PRINT#4,TAB(50)CF$", "DF$
235 I=6:GOSUB1100:C=22
255 FORK=1TOZF
260 PRINT#4,Z$(K)
265 NEXTK:C=C+ZF
300 PRINT#4,TAB(50)L$M$:C=C+1
315 PRINT#4
316 A$=STR$(IV):A$=RIGHT$("   "+A$+" ",4)
320 PRINT#4," I V A  ";A$;"%";TAB(37)L$IV$
325 PRINT#4,TAB(40);S3$:C=C+2
329 PRINT#4,TAB(40)"TOTALE:   "L$TF$
330 I=13:GOSUB1100:C=C+14
335 PRINT#4,"DATI ANAGRAFICI:"
336 PRINT#4,M$(2)" "M$(3)
337 PRINT#4,"NASCITA: "M$(10)
338 PRINT#4,"RESIDENZA: "M$(4)"-"M$(5)"-"M$(6)
339 PRINT#4,C$(8)M$(8)
340 PRINT#4,C$(9)M$(9):C=C+5
341 I=58-C:GOSUB1100
343 A$=M$(4)+"-"+M$(5)+"-"+M$(6)
344 X=LEN(A$):IFX>40THENSTOP
345 X=(80-X*2)/4:IFX=0THEN348
346 A$=LEFT$(S$,X)+A$
348 PRINT#4,CHR$(14)A$CHR$(15)
350 A$="TEL.: "+M$(7):X=LEN(A$)
355 IFX>40THENSTOP
360 X=(80-X*2)/4:IFX=0THEN370
365 A$=LEFT$(S$,X)+A$
```

```
370 PRINT#4,CHR$(14)A$CHR$(15)
375 I=4:GOSUB1100
380 NEXTKF:CLOSE4:STOP
500 MA$=STR$(ZX):X=LEN(MA$)-1
501 IFX<=3THENA$=MA$:GOTO535
505 Z=INT(X/3):Y=X-Z*3:A$=""
510 IFY=0THEN520
515 A$=A$+LEFT$(MA$,Y+1)+"."
520 A$=A$+MID$(MA$,Y+2,3)
525 Z=Z-1:IFZ=0THEN535
530 A$=A$+".":Y=Y+3:GOTO520
535 RETURN
600 K$=STR$(K):IFLEN(K$)=2THENK$=" "+K$
601 RETURN
1000 PRINTV$
1005 INPUT"CONFERMI (S/N):";R$:RETURN
1100 FORK=1TOI:PRINT#4:NEXTK:RETURN
```

Segue una fattura risultato del programma fotografata rimpicciolendola.

SIG. GIULIO BELLINI

SPETT.
DITTA BELLA
VIA CAMPAGNA, 57
110111 COMO

NUMERO FATTURA:39
VOGHERA, 15 GENNAIO 1984

PER LAVORI DI RICERCA DA VOI COMMISSIONATI
COME DA ACCORDI INTERCORSI

L. 600.000

I V A 18 % L. 108.000

TOTALE: L. 708.000

DATI ANAGRAFICI:
GIULIO BELLINI
NASCITA: VERONA-1/1/1947
RESIDENZA: 110111-VOGHERA-VIA SPERANZA, 56
COD. FISCALE: 01000010111
PARTITA IVA: 100010100011

110111-VOGHERA-VIA SPERANZA, 56
TEL.: 1011101011

Nello svolgimento del programma troverai applicati gli argomenti già studiati; in particolare osserva come viene gestito il contatore delle righe di stampa C per poter andare correttamente a nuovo foglio. Inoltre è interessante il sottoprogramma che inizia in 500; esso serve per mettere i puntini separatori nel numero da stampare. Nel rispondere alle richieste di dati, se vuoi che il programma accetti virgole e due punti (per esempio per l'indirizzo, dove si usa far precedere da una virgola il numero civico) devi ricordarti di scrivere il carattere "virgolette" prima del primo carattere valido e subito dopo l'ultimo, cioè scrivere la risposta tra virgolette.
Se vuoi usare il programma per mandare le tue fatture ai clienti principali puoi modificarlo in questo modo:
.. trasformare le linee da 23 a 80 in 10 linee di assegnazione delle 10 stringhe M$, cancellando quelle in più; così i tuoi dati diventano costanti di programma;
.. trasformare le linee da 105 a 131 in 4 linee di assegnazione delle 4 stringhe I$, cancellando quelle in più; così i dati del destinatario diventano costanti di programma;
.. memorizzare una copia del programma su disco o su nastro, assegnandogli un nome di riconoscimento;
.. ripetere le ultime due operazioni per il numero di copie diverse di programma che desideri avere.
In tale modo, per inviare una fattura a un determinato cliente, basta richiamare il relativo programma e introdurre solo i dati della fattura e il suo argomento.

CAPITOLO 4

CODICI E NUMERI DEL CALCOLATORE

4.1 PREMESSA

In un byte si possono rappresentare numeri decimali da 0 a 255.
L'interpretazione che viene data a un byte dipende dal contesto nel quale esso viene preso in esame.
Nell'interpretazione in codice ASCII, una parte di questi 256 numeri corrisponde a caratteri stampabili, una parte degli altri corrisponde a specifiche azioni (codici di controllo), alcuni non hanno effetto.
Il contenuto dei byte può rappresentare istruzioni di programma in linguaggio macchina; in questo caso l'interpretazione viene fatta in un modo particolare e non intendiamo occuparcene in questo libro.
Un byte può contenere un codice che corrisponde a una istruzione del linguaggio Basic.
Uno o più byte associati possono contenere dei numeri espressi in binario.

4.2 NUMERI INTERI

Nell'interpretazione numerico binaria, due byte associati (byte basso + byte alto) contengono un numero intero che, se considerato senza segno, può raggiungere il valore 65535; infatti 255 + 256x255 = 65535. Questo modo è normalmente usato per rappresentare indirizzi di memoria.

Se il numero definito da due byte associati è considerato come numero con segno, allora il primo bit di sinistra del byte alto serve per il segno. Il numero positivo ha questo bit a zero, seguito dai bit che danno il valore assoluto del numero. Il numero negativo ha invece questo bit a uno, seguito dai bit che danno il valore del numero rappresentato in complemento a 2.

Il numero positivo può variare da 0 a 32767; infatti 255 + 256x127 = 32767 (il byte

alto può raggiungere solo il valore 127, dato che il bit di sinistra, bit 7, è usato per il segno). Per esempio:

numero decimale: 35
contenuto dei 2 byte binari: 0000000000100011

numero decimale: 32767
contenuto dei 2 byte binari: 0111111111111111

numero decimale: 0
contenuto dei 2 byte binari: 0000000000000000

Il numero negativo può variare da -32768 a -1. Per arrivare alla rappresentazione binaria di un numero intero negativo si può procedere così:
.. si considera la potenza del 2 immediatamente superiore al massimo numero rappresentabile (con 15 bit a disposizione, 8 del byte basso e 7 del byte alto, si arriva al massimo al valore posizionale di 2 elevato a 14); in questo caso 2 elevato a 15, che è uguale a 32768;
.. si considera il valore assoluto del numero che si vuole rappresentare;
.. si calcola 32768 - (valore assoluto numero);
.. si scrive in binario il risultato del calcolo precedente e si aggiunge un bit 1 a sinistra per il segno.

Esempio: numero decimale da rappresentare = -35
calcolo: 32768 - 35 = 32733
rappresentazione binaria di 32733 ottenuta per sottrazioni successive delle potenze di 2:

32733 -	16384 =	16349	dà un bit 1 in posizione 14
16349 -	8192 =	8157	dà un bit 1 in posizione 13
8157 -	4096 =	4061	dà un bit 1 in posizione 12
4061 -	2048 =	2013	dà un bit 1 in posizione 11
2013 -	1024 =	989	dà un bit 1 in posizione 10
989 -	512 =	477	dà un bit 1 in posizione 9
477 -	256 =	221	dà un bit 1 in posizione 8
221 -	128 =	93	dà un bit 1 in posizione 7
93 -	64 =	29	dà un bit 1 in posizione 6
29 -	16 =	13	dà un bit 1 in posizione 4
13 -	8 =	5	dà un bit 1 in posizione 3
5 -	4 =	1	dà un bit 1 in posizione 2
1 -	1 =	0	dà un bit 1 in posizione 0

il contenuto dei 2 byte risulta quindi, aggiungendo il bit 1 di segno negativo in

posizione15 (si considera posizione 15 quella più a sinistra e posizione 0 quella più a destra):

1111111111011101

Si può arrivare rapidamente allo stesso risultato applicando questa semplice regola:

.. scrivere in binario il valore assoluto del numero, cioè: 35, che equivale a

0000000000100011

.. sostituire ai bit 0 dei bit 1 e ai bit 1 dei bit 0, alla fine sommare un bit 1 nella posizione 0; infatti:

con la sostituzione si ha 1111111111011100
aggiungendo 1 in fondo si ha 1111111111011101.

Se consideriamo il numero -1, otteniamo in binario: 1111111111111111, mentre se consideriamo il numero -32768, otteniamo: 1000000000000000.

Con il programma INTINMEM che segue si stampa sul video e sulla stampante un numero intero con segno e il contenuto dei 7 byte che lo rappresentano in memoria (vedi Capitolo 8).

```
1 REM NUMERI INTERI IN MEMORIA
2 INPUT"□□NUMERO INTERO = ";N%
5 M$="NUMERI INTERI IN MEMORIA"
10 A=256*PEEK(46)+PEEK(45)
15 INPUT"□□VUOI USARE LA STAMPANTE (S/N) ";R$
17 PRINT"□□"M$:PRINT"□N% = ";N%
40 FORK=0TO6:PRINTPEEK(A+K);" ";:NEXTK:PRINT
50 IFR$="N"THEN100
55 OPEN4,4:CMD4:PRINTM$
60 FORK=0TO6:PRINTPEEK(A+K);" ";:NEXTK
70 PRINT:PRINT"N%=";N%:PRINT#4:CLOSE4
100 STOP
```

In questo programma definiamo come prima variabile il numero N% (intero), in modo che si trovi in memoria a partire dalla prima posizione della zona delle variabili (linea 10, calcolo del puntatore A all'inizio della zona variabili). Viene chiesto un numero intero alla linea 2 e poi viene evidenziato il contenuto dei 7 byte che lo contengono in memoria. Il contenuto dei 7 byte è il seguente: •

..primo: il numero 206 che è la somma del codice ASCII di N, cioè il numero 78, con 128 (206=78+128);
..secondo: il numero 128, dato che il nome della variabile è formato da una sola lettera;
..terzo e quarto: il valore del numero in binario;
..quinto, sesto e settimo: zero, dato che il numero rappresentato è intero.

Dagli esempi che seguono puoi verificare quanto abbiamo esposto sopra.

RISULTATI PROGRAMMA PRECEDENTE

```
NUMERI INTERI IN MEMORIA
 206   128   127   255   0   0   0
N%= 32767

NUMERI INTERI IN MEMORIA
 206   128   128   1   0   0   0
N%=-32767

NUMERI INTERI IN MEMORIA
 206   128   0   0   0   0   0
N%= 0

NUMERI INTERI IN MEMORIA
 206   128   255   255   0   0   0
N%=-1
```

4.3 NUMERI REALI

Per rappresentare i numeri reali (non interi, numeri con cifre dopo il punto decimale), che nel calcolatore vengono memorizzati in forma esponenziale, si usano 5 bytes consecutivi. Di questi 5 bytes il primo contiene l'esponente (caratteristica) e gli altri quattro la mantissa. Ricordiamo che un numero in forma esponenziale (floating point) è formato dalle cifre significative del numero (mantissa) e dall'esponente (caratteristica) da usare per calcolare il valore del numero. Nell'aritmetica decimale l'esponente va applicato alla base 10, mentre nell'aritmetica

binaria esso va applicato alla base 2. Il calcolo per ottenere il valore del numero in decimale è il seguente:

numero = mantissa x 10 elevato a esponente.

In questo calcolatore la mantissa viene conservata nella forma: .xxxxxxxxxx (forma normalizzata), considerando solo le cifre significative, e l'esponente ha il valore necessario per calcolare il numero. I limiti in decimale sono: per l'esponente da -39 a +38, e per la mantissa da -42949673 a +42949673. In binario questi limiti diventano per l'esponente da -128 a +127. Per poter gestire un esponente, sia positivo che negativo, usando un solo byte, si usa la convenzione che lo zero per l'esponente è spostato al valore 128; per questa ragione gli esponenti positivi vanno da 128 a 255 e quelli negativi da 0 a 127.
Per calcolare i limiti per la mantissa, si deve considerare il valore ottenibile associando 4 byte consecutivi; si hanno a disposizione 31 bit per il numero e il primo bit a sinistra del byte più alto per il segno. Nel nostro calcolatore, dato che la mantissa del numero viene normalizzata in modo che la prima cifra a sinistra (a destra del punto decimale) è sempre un bit 1, tale bit viene eliminato dalla memoria, ma se ne tiene conto nei calcoli. In conseguenza la mantissa può essere formata da 4 byte contenenti in tutto 32 bit 1. Prima di memorizzare il numero, il primo bit a sinistra viene eliminato e sostituito dal bit del segno, 0 per i numeri positivi e 1 per i numeri negativi. In questo caso, contrariamente a quanto avviene per i numeri interi, i numeri negativi sono memorizzati in valore assoluto, con aggiunto all'inizio il bit 1 di segno.
Si rappresentano in questo modo anche i numeri interi che cadono fuori dell'intervallo —32768/+ 32767.
Nel programma FLOATINMEM che segue, con una tecnica analoga a quella del programma precedente, cioè definendo per prima cosa la variabile N, si ottiene che essa compaia per prima nella zona variabili. Servendosi del puntatore P (calcolato alla linea 4) si leggono dalla memoria con la funzione PEEK i 5 byte che rappresentano il numero; il primo la caratteristica (esponente) e gli altri quattro la mantissa. Il programma stampa il numero introdotto, il contenuto in decimale e in binario dei 4 byte della mantissa, la stringa binaria che costituisce la mantissa come è stata letta inizialmente e quella con aggiunto il primo bit 1 al posto del bit di segno. Viene poi stampato l'esponente come viene letto e come risulta calcolato sottraendo 128, il valore binario della parte intera del numero e il valore binario della parte decimale, ambedue senza segno, e, infine il numero ricalcolato. Puoi fare diverse prove e vedrai che, se superi i limiti consentiti per i numeri, compare una segnalazione di OVERFLOW. In certi casi si hanno differenze tra il numero introdotto e il numero ricalcolato; esse sono dovute ad errori di arrotondamento.
Inoltre i numeri vengono stampati con 9 cifre, mentre in memoria ne sono conservate 10.

```
1 REM CALCOLO NUMERI FLOATING POINT
2 PRINT"[illegible]SCRIVI UN NUMERO DECIMALE":INPUTN
3 IFSW=1THEN8
4 P=PEEK(45)+256*PEEK(46):GOSUB300
5 INPUT"VUOI USARE LA STAMPANTE (S/N) ";R$
6 IFR$="S"THENOPEN4,4:PRINT#4,P$:PRINT#4
8 PRINT"[illegible]"C1$;N:L$=""
9 IFR$="S"THENPRINT#4:PRINT#4,C1$;N
11 IFPEEK(P)<>78THENSTOP
12 FORI=1TO5:X(I)=PEEK(P+I+1):NEXTI
13 PRINTC2$
14 FORI=2TO5:PRINTX(I);"       ";:NEXTI
15 IFR$="N"THEN19
16 PRINT#4,C2$
17 FORI=2TO5:PRINT#4,X(I);"       ";
18 NEXTI:PRINT#4
19 PRINT:PRINTO$
20 IFR$="S"THENPRINT#4,O$
30 FORI=2TO5:GOSUB100:PRINTF$(I);" ";:NEXTI
40 IFR$="N"THEN 60
45 FORK=2TO5:PRINT#4,F$(K);" ";:NEXTK:PRINT#4
60 PRINT
62 PRINTC3$:PRINTL$
63 IFR$="S"THENPRINT#4,C3$:PRINT#4,L$
68 S=1+2*(LEFT$(F$(2),1)="1")
70 F$(2)="1"+RIGHT$(F$(2),7)
72 PRINTN1$:PRINTN2$
73 IFR$="S"THENPRINT#4,N1$:PRINT#4,N2$
75 LL$=F$(2)+F$(3)+F$(4)+F$(5)
76 PRINTLL$
77 IFR$="S"THENPRINT#4,LL$
79 Y=X(1)-128
80 PRINTC8$;X(1)
81 IFR$="N"THEN83
82 PRINT#4,C8$;X(1)
83 PRINTC5$;Y
84 IFR$="S"THENPRINT#4,C5$;Y
85 IFY<0THENM$="0":D$=LEFT$(Z$,-Y)+LL$:GOTO92
86 IFX(1)=0THENM$="0":D$="0":GOTO92
87 IFY>LEN(LL$)THENLL$=LEFT$(LL$+Z$,Y)
89 M$=LEFT$(LL$,Y):Z=LEN(LL$)-Y
90 IFZ<=0THEND$="0":GOTO92
```

```
91 D$=RIGHT$(LL$,(LEN(LL$)-Y))
92 PRINTC6$,M$:PRINTC7$,D$:IFR$="N"THEN95
93 PRINT#4,C6$:PRINT#4,M$
94 PRINT#4,C7$:PRINT#4,D$
95 GOSUB200:GOSUB250:PRINTC4$;(M+D)*S
96 IFR$="S"THENPRINT#4,C4$;(M+D)*S
97 INPUT"ANCORA (S/N) ";B$
98 IFB$="S"THENSW=1:GOTO2
99 CLOSE4:STOP
100 F$(I)="":FORJ=7TO0STEP-1
101 IFINT(X(I)/2↑J)=0THEN105
103 F$(I)=F$(I)+"1":X(I)=X(I)-2↑J:GOTO110
105 F$(I)=F$(I)+"0"
110 NEXT:L$=L$+F$(I):RETURN
156 PRINTD;" ";
200 M=0:IFM$="0"THENRETURN
201 FORJ=LEN(M$)TO1STEP-1
203 IFMID$(M$,J,1)="1"THENM=M+2↑(LEN(M$)-J)
205 NEXTJ:RETURN
250 D=0:IFD$="0"THENRETURN
253 FORJ=1TOLEN(D$)
255 IFMID$(D$,J,1)="1"THEND=D+(1/2)↑J
260 NEXTJ:RETURN
300 N1$="STRINGA BIT CON AGGIUNTO BIT INIZIALE"
301 N2$="AL POSTO DEL BIT DI SEGNO"
302 O$="QUATTRO BYTE MANTISSA IN BINARIO"
303 P$="NUMERI FLOATING POINT"
304 C1$="NUMERO INTRODOTTO = "
306 C2$="QUATTRO BYTE MANTISSA IN MEMORIA"
307 C3$="STRINGA DI BIT INIZIALE"
308 C4$="NUMERO CALCOLATO = "
309 C5$="ESP. PER BASE 2 = "
310 C6$="VAL.BIN.PART.INT. SENZA SEGNO"
311 C7$="VAL.BIN.PART.DEC. SENZA SEGNO"
312 C8$="BYTE ESP. = "
313 Z$="0":FORK=1TO127:Z$=Z$+"0":NEXTK
350 RETURN
```

RISULTATI PROGRAMMA PRECEDENTE

```
NUMERI FLOATING POINT


NUMERO INTRODOTTO =  2.93873588E-39
QUATTRO BYTE MANTISSA IN MEMORIA
 0         0         0         0
QUATTRO BYTE MANTISSA IN BINARIO
00000000 00000000 00000000 00000000
STRINGA DI BIT INIZIALE
00000000000000000000000000000000
STRINGA BIT CON AGGIUNTO BIT INIZIALE
AL POSTO DEL BIT DI SEGNO
10000000000000000000000000000000
BYTE ESP. =  1
ESP. PER BASE 2 = -127
VAL.BIN.PART.INT. SENZA SEGNO
0
VAL.BIN.PART.DEC. SENZA SEGNO
00000000000000000000000000000000000000000000000000
                    00000000000000000000000000000000
00000000000000000000000000000000000000000000000001
                     00000000000000000000000000000000

NUMERO CALCOLATO =  0


NUMERO INTRODOTTO =  4.2949673E+09
QUATTRO BYTE MANTISSA IN MEMORIA
 0         0         0         2
QUATTRO BYTE MANTISSA IN BINARIO
00000000 00000000 00000000 00000010
STRINGA DI BIT INIZIALE
00000000000000000000000000000010
STRINGA BIT CON AGGIUNTO BIT INIZIALE
AL POSTO DEL BIT DI SEGNO
10000000000000000000000000000010
BYTE ESP. =  161
ESP. PER BASE 2 =  33
VAL.BIN.PART.INT. SENZA SEGNO
10000000000000000000000000000100
```

```
VAL.BIN.PART.DEC. SENZA SEGNO
0
NUMERO CALCOLATO =  4.2949673E+09

NUMERO INTRODOTTO =  1E-36
QUATTRO BYTE MANTISSA IN MEMORIA
 42          36          36          154
QUATTRO BYTE MANTISSA IN BINARIO
00101010 00100100 00100100 10011010
STRINGA DI BIT INIZIALE
0010101000100100001001001011010
STRINGA BIT CON AGGIUNTO BIT INIZIALE
AL POSTO DEL BIT DI SEGNO
1010101000100100001001001011010
BYTE ESP. =  9
ESP. PER BASE 2 = -119
VAL.BIN.PART.INT. SENZA SEGNO
0
VAL.BIN.PART.DEC. SENZA SEGNO
0000000000000000000000000000000000000000000000000
                    00000000000000000000000000000000
0000000000000000000000000000000000000000101010100
                                  0100100001001001011010
NUMERO CALCOLATO =  9.991702E-37
```

Come puoi vedere dai risultati riportati, il numero 2.93873588E-39 dà come risultato nel ricalcolo il valore 0, mentre il numero 4.2949673E+09 viene accettato. Ti ricordo che la mantissa fornisce le cifre significative del numero, mentre la caratteristica influisce sull'ordine di grandezza del numero.

4.4 CODICI ASCII E D/CODE - CARATTERI STAMPABILI

Ora ci occupiamo di quello che viene stampato sul video.

Al momento dell'accensione del calcolatore, oppure dopo aver premuto contemporaneamente i tasti RUN/STOP e RESTORE, oppure dopo aver scritto: SYS 64738 seguito da RETURN, le condizioni di colore del video sono:
.. bordo AZZURRO, corrispondente al codice colore 14;
.. sfondo BLU, corrispondente al codice colore 6;
.. cursore AZZURRO, corrispondente al codice colore 14.

Se, in queste condizioni del calcolatore, si vanno a leggere i contenuti dei byte che controllano lo stato dei colori del video si trova:
.. 254 nel byte 53280 che controlla il COLORE DEL BORDO, cioè il codice 14 del colore azzurro più 240;
.. 246 nel byte 53281, che controlla il COLORE DELLO SFONDO, cioè il codice 6 del colore blu più 240;
.. 14 nel byte 646, che controlla il COLORE DEL CURSORE e quindi del CARATTERE, cioè il codice del colore azzurro.

Il calcolatore dispone di due SET di caratteri in codice ASCII; questo codice è usato per la rappresentazione interna dei caratteri. I caratteri sono invece codificati in D/CODE per poter apparire sul video; il D/CODE fa da puntatore alla descrizione del cara.tere per punti.
Ogni SET di caratteri ne comprende 256 (256 codici diversi); di questi 256 caratteri solo 128 sono stampabili, ma in due modi: diretto e inverso, scambiando cioè il colore dello sfondo con quello del carattere (positivo e negativo). Il D/CODE ha quindi 256 codici, 128 per i caratteri diretti e 128 per i caratteri inversi, in ogni set di caratteri.
Sulla tastiera si vedono solo 64 caratteri stampabili, ma essi diventano il doppio usando ogni tasto anche insieme al tasto SHIFT.

La relazione tra codice ASCII e D/CODE è la seguente:

ASCII	D/CODE
32 <= A <= 63	D = A
64 <= A <= 95	D = A - 64
96 <= A <= 127	D = A - 32
160 <= A <= 191	D = A - 64
192 <= A <= 254	D = A - 128
A = 255	D = A - 161

Nelle condizioni di inizializzazione il byte 53272 contiene il numero 21; tale numero serve per selezionare il SET di caratteri MAIUSCOLO/GRAFICO. Se si scrive in questo byte il numero 23 (POKE 53272,23), si ottiene di passare al SET di caratteri MAIUSCOLO/MINUSCOLO. Lo stesso effetto si ottiene premendo contemporaneamente i due tasti COMMODORE e SHIFT. Si può passare da un SET all'altro di caratteri scrivendo:

PRINT CHR$(14) per ottenere il set maiuscolo/minuscolo
PRINT CHR$(142) per ottenere il set maiuscolo/grafico.

In realtà nel byte 53272 il numero 21 e 23 selezionano il set di caratteri, ma anche la posizione di inizio della mappa video a 1024.

I due SET di caratteri (MAPPA CARATTERI) si trovano in ROM dal byte 53248 al byte 57343 e occupano 4096 byte. Infatti per descrivere ogni carattere occorrono 8 byte (8 x 8 punti, 1 bit per punto); 256x8=2048 byte per ogni SET. Quando i bit di posizione 2 e 3 del byte 53272 sono a "10" viene selezionato il SET che inizia a 53248, mentre quando sono a "11" viene selezionato il SET che inizia a 55296.

Con il programma che segue si va a prelevare dalla MAPPA CARATTERI la descrizione di un carattere per punti (bit 0 e 1) e lo si disegna sul video e sulla stampante ponendo al posto di un punto (quello che avviene di norma in modo automatico; 1 bit 1 dà luogo a un punto) un asterisco; inoltre si scrive il valore di ogni byte, che è servito per disegnare il carattere ingrandito, in decimale e in binario.

```
0 REM DISEGNO CARATTERI
1 INPUT"■■RISULTATI SU STAMPANTE? (S/N) ";R$
2 PRINT:IFR$="N"THEN6
3 OPEN4,4:PRINT#4
4 PRINT#4,"***STAMPA IMMAGINE CARATTERI***"
5 PRINT#4
6 D$="":PRINT"QUALE SET VUOI USARE: "
7 PRINT"  1  PER SET MAIUSCOLO/GRAFICO"
8 PRINT"  2  PER SET MAIUSCOLO/MINUSCOLO"
9 INPUT"■■  OPZIONE:";D$:IFD$=""THEN100
10 IFD$<>"1"ANDD$<>"2"THENPRINT"■■■■":GOTO9
14 M$=" SET MAIUSCOLO/MINUSCOLO"
15 IFD$="1"THENM$=" SET MAIUSCOLO/GRAFICO"
16 PRINT"■■SCRIVI IL CODICE DEL CARATTERE"
17 PRINT"IN D/CODE";:INPUTD:PRINT
18 IFD<0ORD>255THENPRINT"■■■■■■":GOTO14
19 POKE56334,PEEK(56334)AND254
20 POKE1,PEEK(1)AND251
21 A=53248+(VAL(D$)-1)*2048
22 FORK=0TO7:D(K)=PEEK(A+D*8+K):NEXTK
23 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
24 PRINT"■■■■D/CODE=";D;M$:PRINT
25 PRINT"CARATTERE CORRISP. AGLI 8 BYTES:"
26 PRINT
27 PRINT"CARATTERE  VAL. BINARIO  VAL. DEC."
28 PRINT:PRINT:IFR$="N"THEN34
29 PRINT#4,"D/CODE=";D;M$:PRINT#4
30 PRINT#4,"CARATTERE CORRISP. AGLI 8 BYTES:"
31 PRINT#4
```

```
32 PRINT#4,"  CARATTERE    VAL. BINARIO   ";
33 PRINT#4,"VAL. DEC.":PRINT#4
34 FORK=0TO7:D$(K)="":E$(K)="":B=D(K)
35 FORJ=0TO7:IFINT(B/2↑(7-J))=0THEN40
36 D$(K)=D$(K)+"*":E$(K)=E$(K)+"1"
39 B=B-2↑(7-J):GOTO41
40 D$(K)=D$(K)+" ":E$(K)=E$(K)+"0"
41 NEXTJ:NEXTK:FORK=0TO7
42 PRINTD$(K);"      ";E$(K);"       ";D(K)
43 IFR$="N"THEN50
44 PRINT#4," ";D$(K);"         ";
45 PRINT#4,E$(K);"       ";D(K)
50 NEXTK
100 IFR$="S"THENCLOSE4
110 STOP
```

RISULTATI DEL PROGRAMMA PRECEDENTE

```
***STAMPA IMMAGINE CARATTERI***

D/CODE= 65  SET MAIUSCOLO/GRAFICO

CARATTERE CORRISP. AGLI 8 BYTES:

  CARATTERE     VAL. BINARIO   VAL. DEC.

     *           00001000        8
    ***          00011100        28
   *****         00111110        62
  *******        01111111        127
  *******        01111111        127
    ***          00011100        28
   *****         00111110        62
                 00000000        0


***STAMPA IMMAGINE CARATTERI***

D/CODE= 193  SET MAIUSCOLO/GRAFICO

CARATTERE CORRISP. AGLI 8 BYTES:

  CARATTERE     VAL. BINARIO   VAL. DEC.

 **** ***        11110111        247
 ***   **        11100011        227
 **     *        11000001        193
 *               10000000        128
 *               10000000        128
 ***   **        11100011        227
 **     *        11000001        193
 ********        11111111        255
```

```
***STAMPA IMMAGINE CARATTERI***

D/CODE= 88  SET MAIUSCOLO/MINUSCOLO

CARATTERE CORRISP. AGLI 8 BYTES:

  CARATTERE     VAL. BINARIO   VAL. DEC.

  **  **          01100110        102
  **  **          01100110        102
   ****           00111100        60
     **           00011000        24
   ****           00111100        60
  **  **          01100110        102
  **  **          01100110        102
                  00000000        0

***STAMPA IMMAGINE CARATTERI***

D/CODE= 193  SET MAIUSCOLO/MINUSCOLO

CARATTERE CORRISP. AGLI 8 BYTES:

  CARATTERE     VAL. BINARIO   VAL. DEC.

 ***  ***        11100111        231
 **     **        11000011        195
 *  **  *        10011001        153
 *      *        10000001        129
 *  **  *        10011001        153
 *  **  *        10011001        153
 *  **  *        10011001        153
 ********        11111111        255
```

Se osservi i risultati precedenti puoi notare che scegliendo lo stesso set di caratteri e usando codici D/CODE che differiscono di 128 si ottiene lo stesso carattere in positivo e negativo.
Il disegno dei caratteri ingranditi corridponde esattamente al disegno dei caratteri per punti sul video. I caratteri stampati dalla stampante differiscono come forma da quelli del video; infatti essi dipendono dalla mappa dei caratteri memorizzata nella ROM della stampante stessa.
Nel programma precedente, la MAPPA CARATTERI è stata prelevata dalla ROM

a partire dall'indirizzo 53248. Per poter operare così si è dovuto sospendere l'uso della tastiera e disattivare le interruzioni automatiche con il primo comando POKE alla linea 19, poi con il secondo comando POKE si è sospeso l'Input/Output normale e resa disponibile la mappa in ROM. A questo punto si sono prelevati i caratteri dalla ROM e poi si sono ripristinate le condizioni normali alla linea 23. Rimandiamo al Capitolo 8 per rivedere l'argomento.
Gli 8 byte che descrivono un carattere sono memorizzati nel vettore D(k). I vettori D(k), D$(k) e E$(k) non sono dimensionati con una DIM, dato che bastano 8 elementi. D$(k) serve per disegnare il carattere con gli asterischi, mentre E$(k) serve per costruire come stringa il valore binario del byte. Nelle righe da 34 a 41 viene fatta la decodifica del valore di ogni byte per cercare i bit 1 presenti; si ottiene questo confrontando la funzione INT del valore del byte diviso per le successive potenze decrescenti di 2, partendo da 2 elevato a 7, con 1 e, nel caso si sia trovato 1, decrementando il byte del valore opportuno.
Nelle condizioni normali la MAPPA VIDEO si trova nelle 1000 locazioni che vanno dal byte 1024 al byte 2023; la MAPPA COLORI si trova nelle 1000 locazioni che vanno dal byte 55796 al byte 56795. Le posizioni di queste due mappe corrispondono ordinatamente alle posizioni del video partendo dall'angolo in alto a sinistra e avanzando carattere per carattere lungo le righe. Affinchè un carattere compaia in una posizione del video, devono essere verificate le seguenti condizioni:
.. nella posizione della MAPPA VIDEO deve essere scritto il codice del carattere in D/CODE;
.. nella posizione corrispondente della MAPPA COLORE deve essere scritto il codice del colore per il carattere, diverso dal colore dello sfondo.
Il sistema va a prelevare dalla tabella descrittiva per punti dei caratteri, tenendo conto del SET selezionato, i bytes che descrivono il carattere e fa comparire i punti che disegnano il carattere sul video. Il D/CODE fa da puntatore per prelevare il carattere nella MAPPA CARATTERI.
La risoluzione del video è in caratteri di 25 righe di 40 caratteri ciascuna; passando ai punti, 8 x 8 per ogni carattere, si ha: (8 x 25) x (8 x 40) = 64000.

Al momento dell'inizializzazione la MAPPA COLORE contiene il codice del colore dello sfondo in tutte le posizioni.

Per stampare sul video si può procedere nei seguenti modi:

1) Usare il comando PRINT seguito dal nome di una variabile numerica o stringa oppure da una costante numerica o stringa; viene evidenziato, carattere per carattere, il contenuto della variabile o la costante, a partire dalla posizione attuale del cursore, nel colore del cursore.

2) Usare il comando PRINT seguito dalla funzione CHR$(x), dove x è il codice ASCII di un carattere stampabile; la stampa avviene con le stesse modalità del punto 1).

3) Scrivere con il comando POKE in una posizione della MAPPA VIDEO il codice D/CODE del carattere e nella corrispondente posizione della MAPPA COLORI il codice del colore desiderato per il carattere (diverso dal codice del colore dello sfondo).

Nei casi 1) e 2) pensano i programmi del sistema a fare quello che fai tu nel caso 3).

Il programma che segue produce una tabella nella quale sono riportati i caratteri stampabili, cioè quelli il cui codice ASCII va da 32 a 127 e da 160 a 255.

```
0 REM CARATTERI E CODICI NEL CBM 64
1 PRINT"■■■■           PROGRAMMA CARATTERI"
2 DIMA(96),D(96),C$(96),I$(96),EA$(96)
3 DIMBA$(96),H$(256),K$(16),IN(96)
4 REM MAPPA VIDEO  DA 1024 A 2023
5 REM MAPPA COLORI DA 55296 A 56295
6 MV=1024:MC=55296
7 REM PUNT. MAPPA VIDEO E MAPPA COLORI
8 GOSUB5000:PRINTCHR$(142):SW=0
9 REM SET MAIUSCOLO/GRAFICO
10 PRINT"■■■      VUOI USARE LA PRINTER S/N ";
11 INPUTR$:PRINT
12 REM STAMPA CARATTERI STAMPABILI
13 REM CODICI ASCII E D/CODE CORRISPONDENTI
14 REM CODICI ASCII DA 32 A 63
15 K1=32:K2=63:PRINT"■";:FORK=K1TOK2
16 PRINTCHR$(K);:NEXTK:PRINT
20 I=0:FORK=K1TOK2
21 POKEMV+32+I,K+128:POKEMC+32+I,1
22 I=I+1:NEXTK
30 GOSUB1100:GOSUB1200:GOSUB1300
40 REM CODICI ASCII DA 64 A 127
45 K1=64:K2=127:GOSUB1000
50 GOSUB1100:GOSUB1200:GOSUB1300
55 REM CODICI ASCII DA 160 A 255
60 K1=160:K2=255:GOSUB1000
65 GOSUB1100:GOSUB1200:GOSUB1300
70 REMSET MAIUSCOLO/MINUSCOLO
80 IFSW=0THENSW=1:PRINTCHR$(14):GOTO14
999 STOP
1000 PRINT"■";:FORK=K1TOK2:PRINTCHR$(K);
1001 NEXTK:PRINTCHR$(18);:FORK=K1TOK2
```

```
1004 PRINTCHR$(K);
1005 NEXTK:PRINTCHR$(146):RETURN
1100 I=1:FORK=K1TOK2:A(I)=K
1101 D(I)=PEEK(MV-1+I)
1103 IN(I)=PEEK(MV-1+I+K2-K1+1)
1105 C$(I)=CHR$(K):I$(I)="▗"+C$(I)+"▝"
1110 EA$(I)=H$(K):H1$=LEFT$(EA$(I),1)
1111 H2$=RIGHT$(EA$(I),1)
1115 H1=ASC(H1$):H2=ASC(H2$)
1116 IFH1<=57THENH1=H1-48:GOTO1118
1117 H1=H1-55
1118 IFH2<=57THENH2=H2-48:GOTO1125
1120 H2=H2-55
1125 BA$(I)=K$(H1)+K$(H2)
1130 I=I+1:NEXTK:RETURN
1200 PRINT"▗":GOSUB2000:I=1
1203 FORK=K1TOK2
1204 IFK<>34THEN1208
1205 PRINTTAB(2)CHR$(34)CHR$(34)"▌ "TAB(6);
1206 PRINT"▗"CHR$(34)CHR$(34)"▝▌ ";
1207 GOTO1210
1208 PRINTTAB(2)C$(I)TAB(6)I$(I);
1210 PRINTTAB(11)A(I)TAB(16)EA$(I);
1213 PRINTTAB(20)BA$(I);
1215 PRINTTAB(30)D(I)TAB(34)IN(I)
1220 FORJ=0TO500:NEXTJ:I=I+1:NEXTK
1221 RETURN
1300 IFR$="N"THENRETURN
1303 OPEN4,4:CMD4
1305 IFSW=0THENPRINTCHR$(145);:GOTO1307
1306 PRINTCHR$(17);
1307 GOSUB2000:I=1:FORK=K1TOK2
1310 PRINTCHR$(16)CHR$(48)CHR$(50)C$(I);
1311 PRINTCHR$(16)CHR$(48)CHR$(54)I$(I);
1315 PRINTCHR$(16)CHR$(48)CHR$(59)A(I);
1317 PRINTCHR$(16)CHR$(49)CHR$(55)EA$(I);
1318 PRINTCHR$(16)CHR$(50)CHR$(50)BA$(I);
1319 PRINTCHR$(16);CHR$(51)CHR$(48)D(I);
1321 PRINTCHR$(16)CHR$(51)CHR$(53)IN(I)
1323 I=I+1:NEXTK
1350 PRINTCHR$(145);:PRINT#4:CLOSE4
1351 RETURN
2000 PRINT"CORRISP.  CARATTERI, CODICI ";
```

```
2001 PRINT"ASCII, D/CODE"
2005 PRINT"I CODICI ASCII SONO IN ";
2006 PRINT"DEC., ESADEC. E BIN."
2010 IFSW=1THEN2015
2011 PRINT"SET MAIUSCOLO/GRAFICO":GOTO2020
2015 PRINT"SET MAIUSCOLO/MINUSCOLO"
2020 PRINT
2025 PRINT"CARATT.                ";
2026 PRINT"ASCII          D/CODE"
2030 PRINT"DIR. INV.  DEC. ";
2031 PRINT"HEX.  BIN.       DEC."
2035 PRINT:RETURN
5000 DATA"000102030405060708090A0B0C0D0E0F"
5001 DATA"101112131415161718191A1B1C1D1E1F"
5003 DATA"202122232425262728292A2B2C2D2E2F"
5004 DATA"303132333435363738393A3B3C3D3E3F"
5005 DATA"404142434445464748494A4B4C4D4E4F"
5006 DATA"505152535455565758595A5B5C5D5E5F"
5007 DATA"606162636465666768696A6B6C6D6E6F"
5008 DATA"707172737475767778797A7B7C7D7E7F"
5009 DATA"808182838485868788898A8B8C8D8E8F"
5010 DATA"909192939495969798999A9B9C9D9E9F"
5011 DATA"A0A1A2A3A4A5A6A7A8A9AAABACADAEAF"
5012 DATA"B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF"
5013 DATA"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF"
5014 DATA"D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF"
5015 DATA"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF"
5016 DATA"F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF"
5017 K=0:FORJ=0TO15:READA$:FORL=1TO16
5018 H$(K)=MID$(A$,L*2-1,2)
5019 K=K+1:NEXTL:NEXTJ
5020 DATA"00000001001000110100010101100111"
5021 DATA"10001001101010111100110111101111"
5022 K=0:FORJ=0TO1:READA$:FORL=0TO7
5023 K$(K)=MID$(A$,(L+1)*4-3,4):K=K+1
5024 NEXTL:NEXTJ
5025 RETURN
```

Nelle prime due colonne della tabella sono riportati i caratteri come appaiono sulla stampante in modo diretto e inverso, nelle tre colonne seguenti sono riportati i codici ASCII in decimale, esadecimale e binario, e nelle ultime due colonne sono riportati i codici decimali D/CODE usati nella mappa video per il carattere diretto e inverso. Il codice del carattere inverso si ottiene aggiungendo 128 al codice del carattere diretto.

RISULTATI DEL PROGRAMMA PRECEDENTE

CARATT.
CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT. DIR.	INV.	ASCII DEC.	HEX.	BIN.	D/CODE DEC.	
		32	20	00100000	32	160
!		33	21	00100001	33	161
"	206	34	22	00100010	34	162
#		35	23	00100011	35	163
$		36	24	00100100	36	164
%		37	25	00100101	37	165
&		38	26	00100110	38	166
'		39	27	00100111	39	167
(		40	28	00101000	40	168
)		41	29	00101001	41	169
*		42	2A	00101010	42	170
+		43	2B	00101011	43	171
,		44	2C	00101100	44	172
-		45	2D	00101101	45	173
.		46	2E	00101110	46	174
/		47	2F	00101111	47	175
0		48	30	00110000	48	176
1		49	31	00110001	49	177
2		50	32	00110010	50	178
3		51	33	00110011	51	179
4		52	34	00110100	52	180
5		53	35	00110101	53	181
6		54	36	00110110	54	182
7		55	37	00110111	55	183
8		56	38	00111000	56	184
9		57	39	00111001	57	185
:		58	3A	00111010	58	186
;		59	3B	00111011	59	187
<		60	3C	00111100	60	188
=		61	3D	00111101	61	189
>		62	3E	00111110	62	190
?		63	3F	00111111	63	191

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT.		ASCII			D/CODE	
DIR.	INV.	DEC.	HEX.	BIN.	DEC.	
@		64	40	01000000	0	128
A		65	41	01000001	1	129
B		66	42	01000010	2	130
C		67	43	01000011	3	131
D		68	44	01000100	4	132
E		69	45	01000101	5	133
F		70	46	01000110	6	134
G		71	47	01000111	7	135
H		72	48	01001000	8	136
I		73	49	01001001	9	137
J		74	4A	01001010	10	138
K		75	4B	01001011	11	139
L		76	4C	01001100	12	140
M		77	4D	01001101	13	141
N		78	4E	01001110	14	142
O		79	4F	01001111	15	143
P		80	50	01010000	16	144
Q		81	51	01010001	17	145
R		82	52	01010010	18	146
S		83	53	01010011	19	147
T		84	54	01010100	20	148
U		85	55	01010101	21	149
V		86	56	01010110	22	150
W		87	57	01010111	23	151
X		88	58	01011000	24	152
Y		89	59	01011001	25	153
Z		90	5A	01011010	26	154
[		91	5B	01011011	27	155
£		92	5C	01011100	28	156
]		93	5D	01011101	29	157
↑		94	5E	01011110	30	158
←		95	5F	01011111	31	159
—		96	60	01100000	64	192
♠		97	61	01100001	65	193
|		98	62	01100010	66	194

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT. DIR.	CARATT. INV.	ASCII DEC.	ASCII HEX.	ASCII BIN.	D/CODE DEC.	
─	▄	99	63	01100011	67	195
─	▄	100	64	01100100	68	196
─	▄	101	65	01100101	69	197
─	▄	102	66	01100110	70	198
│	▌	103	67	01100111	71	199
│	▌	104	68	01101000	72	200
╲	◪	105	69	01101001	73	201
╲	◪	106	6A	01101010	74	202
╲	◪	107	6B	01101011	75	203
└	▄	108	6C	01101100	76	204
╱	◪	109	6D	01101101	77	205
╱	◪	110	6E	01101110	78	206
┌	▄	111	6F	01101111	79	207
┐	▄	112	70	01110000	80	208
●	▐	113	71	01110001	81	209
─	▄	114	72	01110010	82	210
◆	▐	115	73	01110011	83	211
│	▌	116	74	01110100	84	212
╱	◪	117	75	01110101	85	213
╳	▣	118	76	01110110	86	214
○	▣	119	77	01110111	87	215
♠	▣	120	78	01111000	88	216
│	▌	121	79	01111001	89	217
◆	▐	122	7A	01111010	90	218
+	▞	123	7B	01111011	91	219
▒	▒	124	7C	01111100	92	220
│	▌	125	7D	01111101	93	221
π	▞	126	7E	01111110	94	222
◤	◢	127	7F	01111111	95	223
	▄	160	A0	10100000	96	224
▌	▌	161	A1	10100001	97	225
▄	▄	162	A2	10100010	98	226
─	▄	163	A3	10100011	99	227
_	▄	164	A4	10100100	100	228
│	▄	165	A5	10100101	101	229
▒	▒	166	A6	10100110	102	230

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT. DIR.	CARATT. INV.	ASCII DEC.	ASCII HEX.	ASCII BIN.	D/CODE DEC.	
		167	A7	10100111	103	231
		168	A8	10101000	104	232
		169	A9	10101001	105	233
		170	AA	10101010	106	234
		171	AB	10101011	107	235
		172	AC	10101100	108	236
		173	AD	10101101	109	237
		174	AE	10101110	110	238
		175	AF	10101111	111	239
		176	B0	10110000	112	240
		177	B1	10110001	113	241
		178	B2	10110010	114	242
		179	B3	10110011	115	243
		180	B4	10110100	116	244
		181	B5	10110101	117	245
		182	B6	10110110	118	246
		183	B7	10110111	119	247
		184	B8	10111000	120	248
		185	B9	10111001	121	249
		186	BA	10111010	122	250
		187	BB	10111011	123	251
		188	BC	10111100	124	252
		189	BD	10111101	125	253
		190	BE	10111110	126	254
		191	BF	10111111	127	255
		192	C0	11000000	64	192
		193	C1	11000001	65	193
		194	C2	11000010	66	194
		195	C3	11000011	67	195
		196	C4	11000100	68	196
		197	C5	11000101	69	197
		198	C6	11000110	70	198
		199	C7	11000111	71	199
		200	C8	11001000	72	200
		201	C9	11001001	73	201
		202	CA	11001010	74	202

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT. DIR.	CARATT. INV.	ASCII DEC.	ASCII HEX.	ASCII BIN.	D/CODE DEC.	
╱	▟	203	CB	11001011	75	203
╚	▟	204	CC	11001100	76	204
╲	▟	205	CD	11001101	77	205
╱	▟	206	CE	11001110	78	206
╔	▟	207	CF	11001111	79	207
╗	▟	208	D0	11010000	80	208
●	▟	209	D1	11010001	81	209
─	▟	210	D2	11010010	82	210
◆	▟	211	D3	11010011	83	211
│	▟	212	D4	11010100	84	212
╱	▟	213	D5	11010101	85	213
╳	▟	214	D6	11010110	86	214
○	▟	215	D7	11010111	87	215
↑	▟	216	D8	11011000	88	216
│	▟	217	D9	11011001	89	217
◆	▟	218	DA	11011010	90	218
+	▟	219	DB	11011011	91	219
▒	▟	220	DC	11011100	92	220
│	▟	221	DD	11011101	93	221
π	▟	222	DE	11011110	94	222
◢	▟	223	DF	11011111	95	223
	▟	224	E0	11100000	96	224
▮	▟	225	E1	11100001	97	225
▬	▟	226	E2	11100010	98	226
▔	▟	227	E3	11100011	99	227
_	▟	228	E4	11100100	100	228
│	▟	229	E5	11100101	101	229
▒	▟	230	E6	11100110	102	230
│	▟	231	E7	11100111	103	231
▒	▟	232	E8	11101000	104	232
◤	▟	233	E9	11101001	105	233
│	▟	234	EA	11101010	106	234
├	▟	235	EB	11101011	107	235
▪	▟	236	EC	11101100	108	236
└	▟	237	ED	11101101	109	237

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT.		ASCII			D/CODE	
DIR.	INV.	DEC.	HEX.	BIN.	DEC.	
┐	▟	238	EE	11101110	110	238
─	▄	239	EF	11101111	111	239
┌	▛	240	F0	11110000	112	240
┴	▀	241	F1	11110001	113	241
┬	▀	242	F2	11110010	114	242
┤	▜	243	F3	11110011	115	243
│	▌	244	F4	11110100	116	244
▌	▌	245	F5	11110101	117	245
▌	▌	246	F6	11110110	118	246
▔	▄	247	F7	11110111	119	247
▔	▄	248	F8	11111000	120	248
▁	▄	249	F9	11111001	121	249
┘	▄	250	FA	11111010	122	250
▗	▜	251	FB	11111011	123	251
▝	▙	252	FC	11111100	124	252
▟	▟	253	FD	11111101	125	253
▗	▟	254	FE	11111110	126	254
π	▟	255	FF	11111111	94	222

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt.		ascii			d/code	
dir.	inv.	dec.	hex.	bin.	dec.	
	▄	32	20	00100000	32	160
!	▟	33	21	00100001	33	161
"▟06▜"		34	22	00100010	34	162
#	▟	35	23	00100011	35	163
$	▟	36	24	00100100	36	164
%	▟	37	25	00100101	37	165
&	▟	38	26	00100110	38	166
'	▟	39	27	00100111	39	167

corrisP. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	inv.	ascii dec.	hex.	bin.	d/code dec.	
(		40	28	00101000	40	168
)		41	29	00101001	41	169
*		42	2a	00101010	42	170
+		43	2b	00101011	43	171
,		44	2c	00101100	44	172
-		45	2d	00101101	45	173
.		46	2e	00101110	46	174
/		47	2f	00101111	47	175
0		48	30	00110000	48	176
1		49	31	00110001	49	177
2		50	32	00110010	50	178
3		51	33	00110011	51	179
4		52	34	00110100	52	180
5		53	35	00110101	53	181
6		54	36	00110110	54	182
7		55	37	00110111	55	183
8		56	38	00111000	56	184
9		57	39	00111001	57	185
:		58	3a	00111010	58	186
;		59	3b	00111011	59	187
<		60	3c	00111100	60	188
=		61	3d	00111101	61	189
>		62	3e	00111110	62	190
?		63	3f	00111111	63	191
@		64	40	01000000	0	128
a		65	41	01000001	1	129
b		66	42	01000010	2	130
c		67	43	01000011	3	131
d		68	44	01000100	4	132
e		69	45	01000101	5	133
f		70	46	01000110	6	134
g		71	47	01000111	7	135
h		72	48	01001000	8	136
i		73	49	01001001	9	137
j		74	4a	01001010	10	138
k		75	4b	01001011	11	139

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	inv.	ascii dec.	hex.	bin.	d/code dec.	
l		76	4c	01001100	12	140
m		77	4d	01001101	13	141
n		78	4e	01001110	14	142
o		79	4f	01001111	15	143
p		80	50	01010000	16	144
q		81	51	01010001	17	145
r		82	52	01010010	18	146
s		83	53	01010011	19	147
t		84	54	01010100	20	148
u		85	55	01010101	21	149
v		86	56	01010110	22	150
w		87	57	01010111	23	151
x		88	58	01011000	24	152
y		89	59	01011001	25	153
z		90	5a	01011010	26	154
[		91	5b	01011011	27	155
£		92	5c	01011100	28	156
]		93	5d	01011101	29	157
↑		94	5e	01011110	30	158
←		95	5f	01011111	31	159
—		96	60	01100000	64	192
A		97	61	01100001	65	193
B		98	62	01100010	66	194
C		99	63	01100011	67	195
D		100	64	01100100	68	196
E		101	65	01100101	69	197
F		102	66	01100110	70	198
G		103	67	01100111	71	199
H		104	68	01101000	72	200
I		105	69	01101001	73	201
J		106	6a	01101010	74	202
K		107	6b	01101011	75	203
L		108	6c	01101100	76	204
M		109	6d	01101101	77	205
N		110	6e	01101110	78	206
O		111	6f	01101111	79	207

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	caratt. inv.	ascii dec.	ascii hex.	ascii bin.	d/code dec.	
P	▄	112	70	01110000	80	208
Q	▀	113	71	01110001	81	209
R	▄	114	72	01110010	82	210
S	▄	115	73	01110011	83	211
T	▀	116	74	01110100	84	212
U	▀	117	75	01110101	85	213
V	▄	118	76	01110110	86	214
W	▀	119	77	01110111	87	215
X	▄	120	78	01111000	88	216
Y	▄	121	79	01111001	89	217
Z	▄	122	7a	01111010	90	218
+	▄	123	7b	01111011	91	219
¦	▄	124	7c	01111100	92	220
|	▄	125	7d	01111101	93	221
¤	▄	126	7e	01111110	94	222
▒	▒	127	7f	01111111	95	223
	▄	160	a0	10100000	96	224
▌	▌	161	a1	10100001	97	225
▬	▬	162	a2	10100010	98	226
▔	▄	163	a3	10100011	99	227
_	▄	164	a4	10100100	100	228
|	▄	165	a5	10100101	101	229
▒	▒	166	a6	10100110	102	230
|	▄	167	a7	10100111	103	231
▒	▒	168	a8	10101000	104	232
▒	▒	169	a9	10101001	105	233
|	▄	170	aa	10101010	106	234
├	▄	171	ab	10101011	107	235
▪	▄	172	ac	10101100	108	236
└	▄	173	ad	10101101	109	237
┐	▄	174	ae	10101110	110	238
─	▄	175	af	10101111	111	239
r	▄	176	b0	10110000	112	240
┴	▄	177	b1	10110001	113	241
┬	▄	178	b2	10110010	114	242
┤	▄	179	b3	10110011	115	243

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	caratt. inv.	ascii dec.	ascii hex.	ascii bin.	d/code dec.	
		180	b4	10110100	116	244
		181	b5	10110101	117	245
		182	b6	10110110	118	246
		183	b7	10110111	119	247
		184	b8	10111000	120	248
		185	b9	10111001	121	249
		186	ba	10111010	122	250
		187	bb	10111011	123	251
		188	bc	10111100	124	252
		189	bd	10111101	125	253
		190	be	10111110	126	254
		191	bf	10111111	127	255
		192	c0	11000000	64	192
A		193	c1	11000001	65	193
B		194	c2	11000010	66	194
C		195	c3	11000011	67	195
D		196	c4	11000100	68	196
E		197	c5	11000101	69	197
F		198	c6	11000110	70	198
G		199	c7	11000111	71	199
H		200	c8	11001000	72	200
I		201	c9	11001001	73	201
J		202	ca	11001010	74	202
K		203	cb	11001011	75	203
L		204	cc	11001100	76	204
M		205	cd	11001101	77	205
N		206	ce	11001110	78	206
O		207	cf	11001111	79	207
P		208	d0	11010000	80	208
Q		209	d1	11010001	81	209
R		210	d2	11010010	82	210
S		211	d3	11010011	83	211
T		212	d4	11010100	84	212
U		213	d5	11010101	85	213
V		214	d6	11010110	86	214

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	caratt. inv.	ascii dec.	ascii hex.	ascii bin.	d/code dec.	
W	▚	215	d7	11010111	87	215
X	▞	216	d8	11011000	88	216
Y	▚	217	d9	11011001	89	217
Z	▞	218	da	11011010	90	218
┼	▞	219	db	11011011	91	219
▒	▒	220	dc	11011100	92	220
│	▌	221	dd	11011101	93	221
▚	▚	222	de	11011110	94	222
▒	▒	223	df	11011111	95	223
	▌	224	e0	11100000	96	224
▌	▌	225	e1	11100001	97	225
▄	▌	226	e2	11100010	98	226
▔	▌	227	e3	11100011	99	227
_	▌	228	e4	11100100	100	228
_	▌	229	e5	11100101	101	229
▒	▒	230	e6	11100110	102	230
│	▌	231	e7	11100111	103	231
▒	▒	232	e8	11101000	104	232
▒	▒	233	e9	11101001	105	233
│	▌	234	ea	11101010	106	234
├	▞	235	eb	11101011	107	235
▗	▙	236	ec	11101100	108	236
└	▚	237	ed	11101101	109	237
┐	▜	238	ee	11101110	110	238
▝	▌	239	ef	11101111	111	239
┌	▛	240	f0	11110000	112	240
┴	▞	241	f1	11110001	113	241
┬	▞	242	f2	11110010	114	242
┤	▞	243	f3	11110011	115	243
│	▌	244	f4	11110100	116	244
▌	▌	245	f5	11110101	117	245
▌	▌	246	f6	11110110	118	246
▔	▌	247	f7	11110111	119	247
▝	▌	248	f8	11111000	120	248
▗	▌	249	f9	11111001	121	249

```
corrisp.  caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt.                ascii          d/code
dir.  inv.   dec.  hex.   bin.        dec.

 ▗    ▚      250   fa    11111010 122   250
 ▖    ▜      251   fb    11111011 123   251
 ▘    ▙      252   fc    11111100 124   252
 ▝    ▟      253   fd    11111101 125   253
 ▚    ▗      254   fe    11111110 126   254
 ▞    ▖      255   ff    11111111 94    222
```

Il programma usa questa tecnica:

.. pulisce il video e stampa a partire dalla prima posizione con la funzione CHR$(k) un gruppo di caratteri (sottoprogramma in 1000);

.. per il gruppo di caratteri di codice ASCII compreso tra 32 e 63 abbiamo dovuto usare degli accorgimenti per impedire che il carattere "aperte virgolette" disturbasse la stampa;

.. va a leggere con la funzione PEEK (sottoprogramma 1100) dalle posizioni del video, quello che ha stampato e ricava il D/CODE corrispondente ad ogni valore di K usato nella funzione CHR$; inoltre preleva dal vettore H$ il valore esadecimale del codice ASCII e dal vettore K$ il valore binario corrispondente;

.. stampa sul video (sottoprogramma 1200) la tabella ricavata, usando la funzione TAB per ottenere gli incolonnamenti;

.. stampa sulla stampante (sottoprogramma 1300), se abilitata, la tabella usando, invece della funzione TAB, il codice di controllo per gli incolonnamenti: CHR$(16) seguito dai due CHR$ necessari per dare la posizione di stampa cifra per cifra;

.. i 256 codici ASCII in esadecimale sono caricati nel sottoprogramma in 5000 nel vettore H$, ricavandoli dai DATA delle linee 5000-5016, e i 16 valori binari corrispondenti alle 16 cifre esadecimali sono memorizzati nel vettore K$, ricavandoli dai DATA delle linee 5020 e 5021. Nel programma precedente, che stampa il disegno dei caratteri usando gli asterischi, i valori binari sono ottenuti con un procedimento diverso da quello seguito qui. Analogamente i codici ASCII in

esadecimale si possono ottenere con altre sequenze di istruzioni; puoi provare a modificare questo programma;

.. terminata la stampa del set di caratteri maiuscolo/grafico, si passa al set maiuscolo/minuscolo e si esegue nuovamente il programma;

.. per poter ottenere sulla stampante il set maiuscole/minuscole si deve abilitare la stampa del set stesso usando il codice di controllo 17 (PRINT CHR$(17)), mentre per tornare al set maiuscolo/grafico si deve usare il codice di controllo 145 (PRINT CHR$(145));

.. osservando il listato su stampante dei risultati, prima riportato, puoi vedere una irregolarità alla linea che riguarda il codice ASCII 34; questo codice crea un pò di problemi, dato che quando viene stampato ha l'effetto di aprire le virgolette, e questo effetto si trascina e fa stampare anche il carattere di REVERS ON e altri caratteri prima delle altre virgolette. Per risolvere lo stesso problema sul video si è usata una tecnica particolare che puoi rilevare dalle linee 1204-1207 del listato del programma. Sul video si può tornare indietro e cancellare, sulla carta stampata no.

Se osservi con attenzione i risultati stampati dal programma precedente puoi vedere che:

.. i codici ASCII da 192 a 223 danno luogo agli stessi caratteri dei codici da 96 a 127;

.. i codici ASCII da 224 a 254 danno luogo agli stessi caratteri dei codici da 160 a 190;

.. il codice ASCII 255 dà luogo allo stessso carattere del codice 126;

.. nei due set si hanno caratteri identici per i seguenti gruppi di codici:

ASCII	D/CODE
32 - 64	32 - 63 e 0
91 - 96	27 - 64
123 - 125	91 - 94
160 - 168	96 - 104
170 - 185	106 - 121
187 - 191	123 - 127

4.5 CODICI DI CONTROLLO

Ci occupiamo ora dei CODICI DI CONTROLLO, cioè di quei codici che in ASCII corrispondono nei due set di caratteri ai due gruppi di numeri, da 0 a 31 e da 128 a 159. Alcuni di questi codici non hanno significato, e dato che non vengono riconosciuti come caratteri stampabili possono dar luogo ad errori. Questi caratteri non hanno un corrispondente in D/CODE.

I codici di controllo che passiamo ad elencare vengono attivati dal comando PRINT e possono essere passati al sistema come CHR$ o all'interno di una stringa delimitata dalle virgolette (apici, di codice ASCII 34).

CODICE ASCII	AZIONE
5	colore bianco
8	disabilita l'effetto della pressione contemporanea dei tasti SHIFT e COMMODORE se diretto alla stampante attiva il modo grafico per punti
9	abilita l'effetto della pressione contemporanea dei tasti SHIFT e COMMODORE
10	manda un LINE FEED alla stampante
13	corrisponde alla pressione del tasto RETURN sulla stampante produce anche line feed
14	fa passare al SET minuscolo/maiuscolo se diretto alla stampante fa passare a caratteri di doppia ampiezza
15	se diretto alla stampante disabilita l'effetto del codice 14
16	se diretto alla stampante provoca lo spostamento della posizione
17	produce l'effetto del tasto CRSR GIU' se diretto alla stampante attiva il set di caratteri maiuscolo/minuscolo
18	attiva il modo REVERSE ON anche sulla stampante
19	manda il cursore nell'angolo in alto a sinistra
20	attiva la funzione di DELETE
26	se diretto alla stampante fa ripetere i caratteri grafici

27	se diretto alla stampante sposta a una posizione punto specificata
28	colore rosso
29	produce l'effetto del tasto CRSR DESTRA
30	colore verde
31	colore blu
129	colore arancione
133	tasto funzione F1
134	tasto funzione F3
135	tasto funzione F5
136	tasto funzione F7
137	tasto funzione F2
138	tasto funzione F4
139	tasto funzione F6
140	tasto funzione F8
141	produce lo stesso effetto della pressione contemporanea dei tasti SHIFT e RETURN
142	attiva il set maiuscolo/grafico
144	colore nero
145	produce l'effetto del tasto CRSR SU se diretto alla stampante attiva il set maiuscolo/grafico
146	annulla l'effetto di RVS ON corrisponde quindi a RVS OFF anche se diretto alla stampante
147	pulisce il video e manda il cursore nell'angolo in alto a sinistra
148	attiva la funzione di INSERT
149	colore bruno
150	color rosso chiaro
151	colore grigio chiaro
152	colore grigio scuro
153	colore verde chiaro
154	colore blu chiaro
155	colore grigio molto scuro
156	colore porpora
157	produce l'effetto del tasto CRSR SINISTRA
158	colore giallo
159	colore ciano

Come vedi alcuni caratteri agiscono diversamente se diretti con il comando PRINT al video o alla stampante.

Quando ai tasti funzione viene assegnato un compito specifico, il corrispondente codice di controllo lo fa eseguire.

4.6 CODIFICA PAROLE CHIAVE - TOKENS

Le parole chiave del linguaggio Basic si scrivono sulla tastiera carattere per carattere, ma, salvo poche eccezioni, vengono codificate in un byte. Questi codici speciali prendono il nome di TOKENS. Essi coprono l'intervallo dei numeri da 128 a 202, ma non possono essere scambiati con i normali caratteri corrispondenti a questi numeri, dato che vengono interpretati come parole chiave nel contesto nel quale sono presi in esame.

Inoltre, molte parole chiave possono essere digitate sulla tastiera in forma abbreviata; il sistema le riconosce, ma produce sul video una rappresentazione simbolica abbreviata. In memoria vengono memorizzate allo stesso modo di quando si digitano per intero, e, in fase di lista del programma compaiono in forma completa.

Il programma TOKENS che segue produce una tabella, il cui contenuto è il seguente:

.. col. 1) COD.: codici decimali delle parole chiave;

.. col. 2) DES.: parola, o parole chiave, o simboli;

.. col. 3) ABBR.: tasti da digitare sulla tastiera per abbreviare; devi intendere il primo, o i primi due tasti e poi il tasto SHIFT insieme al tasto che segue. Se non esiste abbreviazione, la posizione è in bianco;

.. col. 4) VIDEO: quello che compare sul video per effetto della digitazione abbreviata (di quella normale se non esiste abbreviazione).

```
1 REM TOKENS
5 OPEN4,4:CMD4:GOSUB150:J=128
7 PRINTCHR$(14);"          T O K E N S":PRINT
8 PRINT" COD. DES.      ABBR.      VIDEO":PRINT
9 N=1 :L$=""
10 FORK=41118TO41370
15 L=PEEK(K)
20 IFL<=127THENL$=L$+CHR$(L):NEXTK
25 L=L-128:L$=L$+CHR$(L)
27 L$=LEFT$(L$+S$,7)
```

```
30 PRINTJ;L$;" ";
35 IFA1$(N)<>S$THEN40
37 PRINTS$;"  ";L$:GOTO50
40 PRINTA1$(N);"  ";A2$(N)
50 N=N+1:J=J+1:L$="":NEXTK:PRINTCHR$(15)
100 PRINT#4:CLOSE4:STOP
150 DATA"E N/F O⌐N E¬D A♠I N/    D I↖R E"
151 DATA"¬L E¬G O⌐"
155 DATA"R U/    RES♥GOS♥RET!    S T!    "
156 DATA" W A♠L O⌐"
160 DATA"S A♠V E¬D E¬P O⌐P R_? ? C O⌐L I"
161 DATA"↖C LLC M↖"
165 DATA"S Y IO P¬CLO⌐G E¬    T A♠       "
166 DATA" S P¬T H !"
170 DATA"K O⌐STE¬                    R N"
171 DATA"/         "
175 DATA"         S G!    A BIU S♥F R_   "
176 DATA" S Q♥R N/"
180 DATA"    E X♠    S I↖    A TIP E¬   "
181 DATA" STR_V A♠"
185 DATA"A S♥C H ILEF_R I↖M I↖"
208 DIMA1$(75),A2$(75):S$="          "
209 L=1:FORK=0TO6:READM$:READN$:M$=M$+N$
210 FORJ=0TO9:A1$(L)=MID$(M$,J*4+1,4)
211 L=L+1:NEXTJ:NEXTK
212 READM$:FORJ=0TO4:A1$(L)=MID$(M$,J*4+1,4)
213 L=L+1:NEXTJ:FORK=1TO75
214 A2$(K)=LEFT$(A1$(K),2)
215 A2$(K)=A2$(K)+" "+RIGHT$(A1$(K),1)
217 IFLEFT$(A1$(K),1)<>"?"THEN220
218 A1$(K)="    ?    "
219 A2$(K)="  ? ":GOTO230
220 IFA1$(K)="    "THENA1$(K)=S$:GOTO230
221 N$=LEFT$(A1$(K),2)+" SHIFT "
223 A1$(K)=N$+MID$(A1$(K),3,1)
230 NEXTK
250 RETURN
```

Il programma, dopo aver aperto la stampante alla linea 5, va ad eseguire il sottoprogramma in 150. In esso vengono preparati i due vettori A1$ e A2$ per contenere rispettivamente le colonne 3 e 4 della tabella. I dati per questi vettori sono stati passati con delle istruzioni DATA. Puoi analizzare il sottoprogramma e vedrai

che abbiamo dovuto usare degli accorgimenti per quei codici che non ammettono abbreviazioni. Dopo l'esecuzione del sottoprogramma, viene inizializzato J a 128, codice della parola chiave END.
Alla linea 10 inizia un ciclo FOR per K che varia da 41118 a 41370; questi due numeri sono gli indirizzi che delimitano una zona di memoria ROM del Basic dove sono memorizzate le parole chiave carattere per carattere. La tecnica di memorizzazione è la seguente: una parola chiave viene memorizzata carattere per carattere con il codice ASCII corrispondente; per segnalare la sua fine, viene sommato 128 (80H) al codice dell'ultimo carattere. Nel programma si analizzano i codici letti e quando un codice supera 127 si considera conclusa la parola. Il sistema usa i codici delle parole chiave per puntare alle parole stesse nella tabella ora menzionata; può usare il sistema di creare un puntatore che inizia da 0, se sottrae 128 al codice, oppure che inizia da 1, se sottrae 127. Per il momento non siamo andati a vedere come il sistema agisce, ma quello che importa e che tu ti impadronisca di questi tipi di tecniche di programmazione.

RISULTATI PROGRAMMA TOKENS

T O K E N S

COD.	DES.	ABBR.			VIDEO
128	END	E	SHIFT	N	E ⁄
129	FOR	F	SHIFT	O	F ⌐
130	NEXT	N	SHIFT	E	N ¯
131	DATA	D	SHIFT	A	D ♠
132	INPUT#	I	SHIFT	N	I ⁄
133	INPUT				INPUT
134	DIM	D	SHIFT	I	D ~
135	READ	R	SHIFT	E	R ¯
136	LET	L	SHIFT	E	L ¯
137	GOTO	G	SHIFT	O	G ⌐
138	RUN	R	SHIFT	U	R ⁄
139	IF				IF
140	RESTORE	RE	SHIFT	S	RE ♥
141	GOSUB	GO	SHIFT	S	GO ♥
142	RETURN	RE	SHIFT	T	RE |
143	REM				REM
144	STOP	S	SHIFT	T	S |
145	ON				ON
146	WAIT	W	SHIFT	A	W ♠
147	LOAD	L	SHIFT	O	L ⌐
148	SAVE	S	SHIFT	A	S ♠
149	VERIFY	V	SHIFT	E	V ¯
150	DEF	D	SHIFT	E	D ¯
151	POKE	P	SHIFT	O	P ⌐
152	PRINT#	P	SHIFT	R	P _

153	PRINT		?			?
154	CONT	C	SHIFT	O	C	┌
155	LIST	L	SHIFT	I	L	┐
156	CLR	C	SHIFT	L	C	└
157	CMD	C	SHIFT	M	C	┘
158	SYS	S	SHIFT	Y	S	│
159	OPEN	O	SHIFT	P	O	┬
160	CLOSE	CL	SHIFT	O	CL	┴
161	GET	G	SHIFT	E	G	─
162	NEW				NEW	
163	TAB<	T	SHIFT	A	T	♠
164	TO				TO	
165	FN				FN	
166	SPC(	S	SHIFT	P	S	┐
167	THEN	T	SHIFT	H	T	│
168	NOT	N	SHIFT	O	N	┴
169	STEP	ST	SHIFT	E	ST	─
170	+				+	
171	-				-	
172	*				*	
173	/				/	
174	↑				↑	
175	AND	A	SHIFT	N	A	╱
176	OR				OR	
177	>				>	
178	=				=	
179	<				<	
180	SGN	S	SHIFT	G	S	│
181	INT				INT	
182	ABS	A	SHIFT	B	A	│
183	USR	U	SHIFT	S	U	♥
184	FRE	F	SHIFT	R	F	─
185	POS				POS	
186	SQR	S	SHIFT	Q	S	●
187	RND	R	SHIFT	N	R	╱
188	LOG				LOG	
189	EXP	E	SHIFT	X	E	♠
190	COS				COS	
191	SIN	S	SHIFT	I	S	╲
192	TAN				TAN	
193	ATN	A	SHIFT	T	A	│
194	PEEK	P	SHIFT	E	P	─
195	LEN				LEN	
196	STR$	ST	SHIFT	R	ST	─
197	VAL	V	SHIFT	A	V	♠
198	ASC	A	SHIFT	S	A	♥
199	CHR$	C	SHIFT	H	C	│
200	LEFT$	LE	SHIFT	F	LE	─
201	RIGHT$	R	SHIFT	I	R	╲
202	MID$	M	SHIFT	I	M	╲

Come puoi vedere la tabella è stata stampata con caratteri ingranditi, usando il codice di controllo 14.

Dalla tabella precedente mancano le seguenti parole chiave:

TIME TIME$ STATUS π GET#

le prime tre sono abbreviate abitualmente in TI, TI$ e ST.

Per vedere come esse sono rappresentate in memoria, abbiamo scritto un programma, nel quale sono usate queste parole chiave, e poi, il programma stesso, lista la sua rappresentazione in memoria.

```
1 REM ALTRITOKENS
2 GOTO100
10 PRINTTI
20 PRINTTI$
30 PRINTST
40 PRINTπ
50 GET#3,A$
60 STOP
100 A=PEEK(43)+256*PEEK(44)
105 OPEN4,4:CMD4
107 PRINT"LISTA BYTE PROGRAMMA"
110 PRINT:K=0
120 FORJ=1TO4:X=PEEK(A+K):PRINTX;
123 K=K+1:NEXTJ
125 PRINT:N=0:PRINT"    ";
126 X=PEEK(A+K):PRINTX;:N=N+1
127 IFX=0THEN130
128 IFN>8THENPRINT:N=0:PRINT"    ";
129 K=K+1:GOTO126
130 PRINT:IFPEEK(A+K+1)=0THEN160
150 K=K+1:GOTO120
160 PRINT#4:CLOSE4:STOP
```

Se osservi il listato vedi che il programma contiene un errore; infatti alla linea 50 si opera con una GET#, su un canale non aperto. Per questa ragione le istruzione dalla linea 10 alla linea 60 non vengono mai eseguite, ma sono state scritte solo per analizzare la loro memorizzazione. Alla linea 2, con GOTO 100, si va ad eseguire la routine che stampa le istruzioni byte dopo byte, come si trovano in memoria. La

routine da 100 a 160 può essere sfruttata come sottoprogramma sostituendo allo STOP finale un RETURN, ed essere usata in altri programmi. Essa sfrutta il fatto che il programma termina con un byte contenente zero, subito dopo lo zero finale dell'ultima istruzione. Il listato riporta su una riga i due byte che formano il LINK (puntano cioè all'indirizzo della istruzione seguente) e i due byte che danno il numero di linea Basic. Nella riga successiva è listata l'istruzione, andando a capo dopo 9 byte e alla fine della istruzione.
Andiamo a vedere la linea che inizia con: 36 8 10 0; essa è la linea di programma 10. Il 153 che segue è il codice di PRINT, 84 e 73 sono i codici ASCII delle lettere T e I. Nella istruzione seguente vediamo che per TI$ sono presenti i codici: 84, 73 e 36, corrispondenti ai tre caratteri della parola.
Nella istruzione seguente vediamo che sono presenti i codici di S e di T: 83 e 84. Proseguendo, troviamo per π il codice 255.
Infine per GET# troviamo 161, codice di GET, seguito da 35, codice di #.
Puoi proseguire nell'analisi di questo tabulato e soddisfare alcune curiosità, sempre che la cosa ti interessi.

RISULTATI PROGRAMMA PRECEDENTE

```
LISTA BYTE PROGRAMMA

19  8  1  0
   143  32  65  76  84  82  73  84  79
   75  69  78  83  0
28  8  2  0
   137  49  48  48  0
36  8  10  0
   153  84  73  0
45  8  20  0
   153  84  73  36  0
53  8  30  0
   153  83  84  0
60  8  40  0
   153  255  0
71  8  50  0
   161  35  51  44  65  36  0
77  8  60  0
   144  0
99  8  100  0
   65  178  194  40  52  51  41  170  50
   53  54  172  194  40  52  52  41  0
111  8  105  0
```

```
   159  52  44  52  58  157  52  0
139  8  107  0
   153  34  76  73  83  84  65  32  66
   89  84  69  32  80  82  79  71  82
   65  77  77  65  34  0
149  8  110  0
   153  58  75  178  48  0
173  8  120  0
   129  74  178  49  164  52  58  88  178
   194  40  65  170  75  41  58  153  88
   59  0
186  8  123  0
   75  178  75  170  49  58  130  74  0
204  8  125  0
   153  58  78  178  48  58  153  34  32
   32  32  34  59  0
227  8  126  0
   88  178  194  40  65  170  75  41  58
   153  88  59  58  78  178  78  170  49
   0
240  8  127  0
   139  88  178  48  167  49  51  48  0
7  9  128  0
   139  78  177  56  167  153  58  78  178
   48  58  153  34  32  32  32  34  59
   0
22  9  129  0
   75  178  75  170  49  58  137  49  50
   54  0
44  9  130  0
   153  58  139  194  40  65  170  75  170
   49  41  178  48  167  49  54  48  0
59  9  150  0
   75  178  75  170  49  58  137  49  50
   48  0
71  9  160  0
   152  52  58  160  52  58  144  0
```

E' evidente che le routine del sistema quando incontrano la parola chiave GET, vanno ad analizzare il carattere successivo per stabilire se è un #; inoltre ci devono essere dei controlli per stabilire se sono in presenza delle altre quattro parole chiave sopra citate.

3

INDICE

Capitolo 1 - ERRORI
1.1 Tipi di errori 1
1.2 Ricerca errori nei programmi 2
1.3 Errori segnalati dal sistema 3

Capitolo 2 - IL VIDEO E I CARATTERI
2.1 La tastiera 7
2.2 L'interfaccia calcolatore-utente 13
2.2.1 Screen Editor 13
2.2.2 Buffer della tastiera (Keyboard Queue) 13
2.2.3 Modo tra virgolette (Quote Mode) 13
2.2.4 Modo inserimento (Insert Mode) 14
2.2.5 Il codice ASCII-CBM 15
2.2.6 Organizzazione dello schermo e Editor 16
2.2.7 Riferimenti in pagina zero 19
2.3 Il video 21
2.3.1 Display Code 21
2.3.2 Colore caratteri, sfondo e bordo 22
2.3.3 Visualizzazione 23
2.4 Lavorare in modo caratteri 26
2.4.1 Input controllato 26
2.4.2 Maschere video 35
2.4.3 Presentazione dei dati 36
2.5 Grafica in modo caratteri 41
2.5.1 Grafica a bassissima risoluzione 41
2.5.2 Grafica a bassa risoluzione 50
2.5.3 Istogrammi 56
2.6 Esempi di uso dei caratteri grafici 60
2.7 Colloquio sullo schermo 80
2.7.1 Il linguaggio naturale - I sottolinguaggi 80
2.7.2 I menù 81
2.7.3 Come evidenziare una scelta 83

Capitolo 3 - LA TASTIERA
3.1 La tastiera 85

Capitolo 4 - IL BASIC COMPILATO
4.1 Il Basic compilato 89

CAPITOLO 1

ERRORI

1.1 TIPI DI ERRORI

Scrivendo programmi in Basic per il calcolatore si possono commettere diversi tipi di errori.
Gli errori più semplici da correggere sono in generale quelli per i quali il sistema invia un messaggio di segnalazione.
In questa categoria rientrano gli errori di sintassi che commettiamo nello scrivere le frasi del linguaggio. Questo Basic accetta quando si lavora sotto EDITOR le frasi errate, le memorizza, si accorge che sono errate quando tenta di eseguirle. Comunque questi errori sono facili da correggere, al limite andiamo a rivedere sul manuale la sintassi corretta e sistemiamo.
Altri errori che vengono segnalati dal sistema durante l'esecuzione del programma sono quelli che nascono da dati errati e non adatti al tipo di operazione che si vuole eseguire. Tipico errore di questa categoria è quello che si ha quando in un calcolo si imposta una divisione con divisore variabile e questo diventa zero, oppure quando si estrae la radice (funzione SQR) da una espressione che assume valore negativo. Altro errore di questa categoria è arrivare ad usare degli indici che vanno fuori dal range stabilito con la DIM. Oppure scrivere male un file di dati e poi in lettura avere segnalazione di errore. Questi tipi di errori possono essere difficili da correggere se il programma è costruito usando algoritmi complessi.

Tra i messaggi del sistema ci sono anche quelli "non di errore", ma di "situazione", come: BREAK IN LINE, che qui non consideriamo.

Quando abbiamo parlato dei numeri del calcolatore abbiamo fatto notare che si possono avere errori di arrotondamento nei calcoli; devi fare attenzione e usare prudenza nello stabilire confronti tra i risultati dei calcoli.

Un discorso a parte meritano gli errori di logica che commettiamo quando impostiamo un programma per il calcolatore. Si verifica la situazione che il programma "gira", cioè arriva fino in fondo, ma i risultati che ci fornisce sono molto diversi da

quello che ci aspettavamo. Potrebbe anche succedere che il programma è giusto, i ragionamenti di base per la sua impostazione sono giusti, ma noi abbiamo sbagliato a preparare i risultati dei casi prova! A me questo caso non si è mai presentato, ma a qualcuno sarà sicuramente successo.
Comunque questi tipi di errori sono difficili da sistemare; dobbiamo rivedere tutta l'impostazione del programma e ripercorrere passo a passo il lavoro fatto. Sono circostanze nelle quali, se non si è proceduto in modo ordinato documentando la procedura, si è portati a dare in escandescenze verbali. Se siamo saggi, la prossima volta lavoreremo meglio.

1.2 RICERCA ERRORI NEI PROGRAMMI

Per nostra fortuna il Basic facilita la ricerca degli errori nei programmi. Infatti possiamo porre in una prima stesura parecchi STOP nel programma; ogni volta che durante l'esecuzione viene incontrato uno STOP, il programma si arresta ed esce sul video il messaggio BREAK IN LINE. In tale modo sappiamo dove siamo, possiamo consultare il listato del programma e leggere in modo immediato i contenuti delle variabili che ci interessano al momento. Attenzione però a non entrare in EDITOR, in tale caso si perdono i contenuti delle variabili e bisogna ricominciare da capo.
Quando siamo sicuri che una parte di programma va bene possiamo eliminare gli STOP che sono serviti solo in fase di prova. Per riconoscere facilmente le frasi da togliere dopo le prove possiamo inserire dopo le istruzioni una REM con dei caratteri facilmente riconoscibili sul listato.
Oltre agli STOP, possiamo inserire nel programma delle istruzioni che stampino sul video risultati intermedi nei punti più delicati del programma prima della fermata. In tale modo non è necessario intervenire in modo immediato per ricercare risultati. Ricordati che dopo uno STOP il programma prosegue se si scrive CONT e si preme RETURN.
Ricordati anche che è meglio avere sempre a portata di mano un listato del programma.

La tecnica dei sottoprogrammi aiuta a scrivere programmi senza errori. Infatti un sottoprogramma svolge in generale un compito ben determinato ed è formato da poche linee di programma facilmente provabili. Per provare un sottoprogramma bastano poche linee di programma principale che lo lanciano e stampano i risultati ottenuti.
E' consigliabile scrivere per prima cosa i sottoprogrammi e provarli usando poche linee di lancio che poi si eliminano facilmente dal programma definitivo.
Risulta molto più difficile provare un programma lungo in un colpo solo; se ci sono errori, può essere difficile localizzarli.

1.3 ERRORI SEGNALATI DAL SISTEMA

Passiamo ad elencare in ordine alfabetico i messaggi del sistema.

.. BAD DATA
Il programma attende da un file aperto su una priferica dati numerici, arrivano invece dati non numerici.

.. BAD SUBSCRIPT
Viene richiamata una variabile con indice che non corrisponde al dimensionamento della stessa.

.. CAN'T CONTINUE
In seguito al comando CONT il programma non può proseguire per una delle seguenti ragioni:
.. non esiste un programma in memoria,
.. si è entrati in Editor e quindi si è perso il riferimento per continuare,
.. il programma non ha ricevuto il comando RUN,
.. è stato segnalato un errore.

.. DEVICE NON PRESENT
La periferica richiesta per una operazione del tipo: OPEN, CLOSE, CMD, INPUT #, GET #, PRINT #, non è presente (può essere solo spenta).

.. DIVISION BY ZERO
Si è tentato di dividere per zero e questo non è consentito.

.. EXTRA IGNORED
Sono stati scritti più dati di quanti richiesti in risposta a un comando di INPUT. Sono accettati tanti dati quante sono le variabili presenti nella lista, gli altri vanno persi, ma il programma continua.

.. FILE NOT FOUND
Non si trova il file sulla periferica: per la cassetta si è arrivati alla segnalazione di fine nastro senza trovare il file richiesto, per il disco il nome del file non compare nell'indice del disco.

.. FILE NOT OPEN
Si cerca di lavorare con istruzioni del tipo: CLOSE, CMD, INPUT #, GET #, PRINT #, su un file che non è stato aperto.

.. FILE OPEN
Si tenta di aprire un file già aperto.

.. FORMULA TOO COMPLEX
Si è usata una espressione di calcolo troppo complicata e l'interprete Basic non riesce a calcolarla. Conviene spezzare l'espressione in due parti e riprovare, o calcolare prima separatamente qualche parentesi e sostituire il nome di una variabile che contiene un risultato parziale.

.. ILLEGAL DIRECT
Si è tentato di usare in modo immediato un comando che non può essere usato in tale modo, come GET, INPUT, DEFFN.

.. ILLEGAL QUANTITY
Si è tentato di usare una funzione con un argomento fuori dai limiti consentiti o che non rispetta le regole inerenti alla funzione. Nella spiegazione delle frasi Basic abbiamo citato spesso questo tipo di errore.

.. LOAD
Durante il caricamento di un programma da nastro si trovano errori sulla prima e/o sulla seconda registrazione.

.. NEXT WITHOUT FOR
Si incontra un NEXT senza aver prima incontrato FOR, oppure la variabile del NEXT non corrisponde a quella dell'ultimo FOR aperto.

.. NOT INPUT FILE
Si tenta di leggere da un file che è stato aperto per scrivere.

.. NOT OUTPUT FILE
Si tenta di scrivere su un file che è stato aperto per leggere oppure che può essere solo di lettura, come la tastiera. Il file Video non dà questo tipo di errore.

.. OUT OF DATA
Si tenta di leggere con READ più dati di quelli disponibili. Si ha questo errore se si preme RETURN sopra il READY. del video, infatti viene interpretato come READ Y.

.. OUT OF MEMORY
Manca memoria per lavorare; le cause possono essere:
.. .si sta scrivendo un programma troppo lungo;
.. .in fase esecutiva le stringhe sono diventate molto lunghe ed è finita la RAM disponibile per i corpi delle stringhe;
.. .non si fa un uso corretto dei cicli FOR/NEXT (non si chiudono bene) o se ne concatenano più di 9;
.. .non si fa un uso corretto delle istruzioni GOSUB/RETURN, per esempio si eseguono dei GOSUB, ma poi mancano i RETURN.

Nei primi due casi manca la RAM disponibile per il Basic, negli ultimi due manca spazio nell'area STACK di lavoro del sistema. Per stabilire di cosa si tratta basta scrivere in immediato:

PRINT FRE(0) e poi RETURN

se il numero che esce non è zero, significa che si è in una delle due ultime circostanze e ci sono errori nella costruzione del programma.

.. OVERLOW
Si sono eseguiti calcoli il cui risultato supera le capacità del calcolatore.

.. REDIM'D ARRAY
Si tenta di usare la istruzione DIM per un ARRAY che è già stato dimensionato.

.. REDO FROM START
Si è risposto a una richiesta di dati numerici con caratteri non validi. Dopo il messaggio ricompare il punto interrogativo di richiesta e il programma attende i dati di nuovo prima di proseguire.

.. RETURN WITHOUT GOSUB
Si incontra un RETURN senza aver prima eseguito un GOSUB.

.. STRING TOO LONG
Si è cercato di sommare delle stringhe e si superano in totale 255 caratteri; oppure si legge da file una stringa con più caratteri di quelli consentiti per INPUT.

.. SYNTAX
Una istruzione non è riconoscibile, può contenere i più diversi errori di scrittura.

.. TYPE MISMATCH
Si usa un tipo di dato errato, stringa al posto di numero o viceversa.

.. UNDEF'D FUNCTION
Si richiama una funzione che non è stata definita.

.. UNDEF'D STATEMENT
Una istruzione cita un numero di linea inesistente nel programma.

.. VERIFY
L'operazione di verifica per un programma su nastro o su disco non dà buon risultato.

I messaggi REDO FROM START e EXTRA IGNORED sono gli unici che non interrompono il programma; essi hanno solo una funzione di segnalazione.

CAPITOLO 2

IL VIDEO E I CARATTERI

2.1 LA TASTIERA

Quando accendi il tuo COMMODORE 64, se tutti i collegamenti sono stati effettuati nel modo corretto, appare sul video il messaggio iniziale seguito dal quadratino lampeggiante, il CURSORE, che ti segnala la disponibilità del calcolatore ad accettare comandi e programmi in BASIC. Nel seguito, salvo avviso contrario, ci riferiamo a un COMMODORE 64 senza alcuna modifica o aggiunta, nè RAM nè ROM.
Per comunicare con il calcolatore usi il più tipico dei dispositivi di ingresso per dati alfanumerici: la tastiera.
Quella del COMMODORE 64 è una tastiera AMERICANA tipo QWERTY (infatti la prima riga di tasti alfabetici comincia con Q, W, E, R, T, Y, mentre nelle tastiere italiane la Z sostituisce la W, dando QZERTY) da 62 tasti, affiancata a una colonna di altri 4 tasti. Nella Figura 2.1 è riportato uno schema della tastiera.

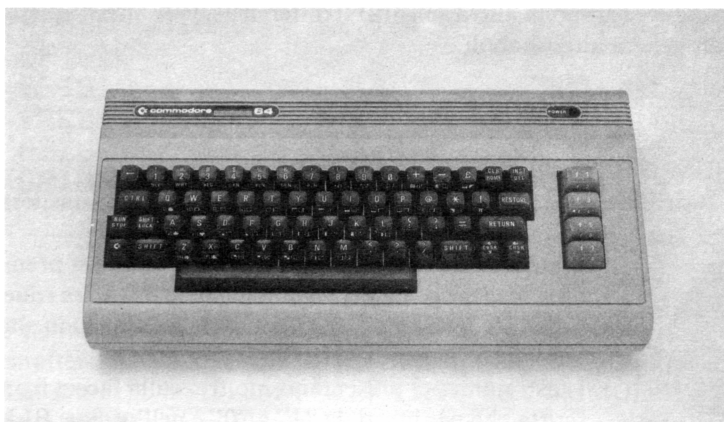


Figura 2.1 Tastiera del COMMODORE 64

La pressione di ciascuno di questi tasti produce un effetto diverso che, se il cursore lampeggiante è presente sullo schermo, viene immediatamente visualizzato. In realtà la pressione di un tasto provoca ben più di un effetto. Questo lo vedremo più avanti. Nella tastiera possiamo distinguere 4 gruppi di tasti:

- alfabetici,
- interpunzione,
- numerici,
- funzione.

TASTI ALFABETICI

Sono 32 e includono, oltre le 26 lettere, anche i simboli:

"@" "*" "↑" "+" "–" "£"

Ad eccezione di" ↑"tutti questi tasti hanno l'aspetto simile a quello mostrato a lato. Nelle condizioni dell'accensione, quando premi il tasto viene generato il simbolo riportato sulla parte alta del tasto ("A"). Il simbolo di destra ("♠") lo ottieni premendo il tasto assieme ad uno dei due SHIFT, posti nell'ultima linea, ai lati della tastiera (corrispondono ai tasti per le maiuscole nelle macchine da scrivere). Il simbolo a sinistra (" r") lo ottieni premendo assieme al tasto quello che reca il simbolo (logo) della COMMODORE, ultimo tasto posto in basso a sinistra sulla tastiera. D'ora in poi ci riferiremo ad esso come "tasto CBM".
Sulla parte frontale del tasto" ↑"compare il simbolo"π". Lo si ottiene con l'uso di SHIFT
Il tasto "←" (primo in alto a sinistra) si differenzia dagli altri di questo gruppo poichè non genera altri simboli.

TASTI NUMERICI E DI INTERPUNZIONE

Sono 10, tutti sulla prima fila, più altri 5 posti sulla destra della tastiera.

Il loro aspetto è mostrato a fianco. Quando si preme il tasto da solo viene generato il simbolo più in basso tra i due presenti sulla faccia superiore ("1"), mentre quello più in alto ("!") si ottiene con l'uso di SHIFT.
I tasti numerici presentano inoltre, sulla faccia frontale una scritta che per i tasti da "1" a "0" è nell'ordine: BLK, WHT, RED, CYN, PUR, GRN, BLU, YEL, RVS ON, RVS OFF. Tali scritte sono delle

abbreviazioni mnemoniche dei termini inglesi che descrivono l'effetto provocato dall'uso contemporaneo di CTRL e del tasto, che può essere:

- la generazione del codice di colore del cursore, che viene resa immediatamente visibile sullo schermo causando:
  - il cambio immediato del colore del cursore,
  - oppure la comparsa di un carattere di controllo, in campo inverso, corrispondente al colore selezionato.

L'uso contemporaneo del tasto CBM e di uno dei tasti da 1 a 8 genera altri otto codici di colore del cursore. Nella versione EXECUTIVE del COMMODORE 64 sui tasti numerici compaiono anche le indicazioni mnemoniche di questi colori. Lo schema a cui puoi riferirti è riportato nella Tabella 2.1, e potrà esserti utile per decifrare listati di programmi ottenuti con una stampante predisposta per il COMMODORE 64.

TABELLA CORRISPONDENZA COLORI

COLORE	CTRL+	CBM+	CAR.	CHR$
NERO	1		"■"	144
BIANCO	2		"■"	5
ROSSO	3		"■"	28
CIANO	4		"■"	159
PORPORA	5		"■"	156
VERDE	6		"■"	30
BLU	7		"■"	31
GIALLO	8		"■"	158
ARANCIONE		1	"■"	129
MARRONE		2	"■"	149
ROSSO-CHIARO		3	"■"	150
GRIGIO-1		4	"■"	151
GRIGIO-2		5	"■"	152
VERDE-CHIARO		6	"■"	153
AZZURRO		7	"■"	154
GRIGIO-3		8	"■"	155

Tabella 2.1 Corrispondenza COLORI-TASTI-CARATTERI-CHR$

A questo punto vogliamo farti notare due cose:

1) il carattere REVERSE ON si può ottenere anche con CTRL-R. Questo dipende dal fatto che il SISTEMA OPERATIVO del COMMODORE 64 è stato direttamente derivato da quello dei PET/CBM, calcolatori nati per l'ufficio e la gestione, come quelli della serie 2000, 3000, 4000 e 8000.

2) Il carattere SPAZIO si ottiene premendo la barra spaziatrice. L'uso contemporaneo di uno SHIFT non provoca alcuna differenza visibile, ma il SISTEMA OPERATIVO è perfettamente in grado di notare la differenza tra SPAZIO e SHIFT-SPAZIO.

TASTI DI COMANDO/SERVIZIO
Sono 12 in tutto e passiamo ad esaminarli.

Il tasto CTRL, oltre alla selezione del colore e del modo diretto o inverso del cursore, se premuto durante l'esecuzione di un LISTing o di una stampa da programma, rallenta lo scrolling del video. Se premuto da solo non genera carattere di controllo.

Il tasto RUN/STOP ha tre funzioni fondamentali:

1) premuto da solo permette di interrompere l'esecuzione di un programma BASIC o di un'operazione di sistema, come lettura o scrittura su nastro o altro dispositivo.

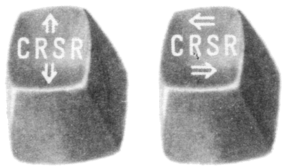
2) premuto contemporaneamente a uno SHIFT produce la sequenza di comandi:
LOAD + RETURN + RUN + RETURN nel COMMODORE 64 standard che carica e lancia il primo programma presente su nastro,
LOAD "*",8 + RETURN + RUN + RETURN nella versione EXECUTIVE che carica e lancia il primo programma presente sul dischetto.

3) premuto contemporaneamente al tasto RESTORE (che genera un segnale riconosciuto dal microprocessore del COMMODORE 64), ripristina (quando possibile) le condizioni iniziali del calcolatore, senza cancellare il programma BASIC, eventualmente presente in memoria.

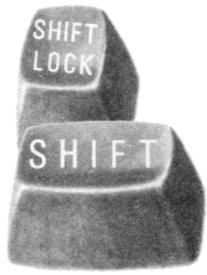
Il tasto CLR/HOME serve per riportare il cursore nell'angolo in alto a sinistra dello schermo. Premuto insieme a uno SHIFT provoca anche la completa pulizia dello schermo.

Il tasto INST/DEL permette di correggere ciò che si sta scrivendo sullo schermo, CANCELLANDO, se usato da solo, il carattere precedente e spostando indietro di un carattere

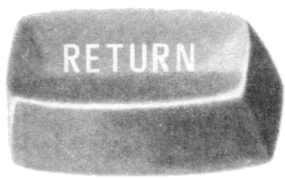
anche il resto della linea sulla quale si trova il cursore; INSERENDO, se usato insieme a uno SHIFT, finchè possibile, uno spazio nella posizione occupata dal cursore e spostando a destra di una posizione il resto della linea logica.



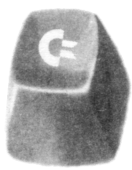
I due tasti CRSR servono per spostare il cursore sullo schermo, orizzontalmente o verticalmente; in giù o a destra, se usati da soli, in su o a sinistra se usati insieme a uno SHIFT.



Sulla tastiera compaiono due tasti SHIFT, uno a sinistra e uno a destra, più un tasto SHIFT/LOCK, che, premuto una volta BLOCCA la funzione degli SHIFT su ATTIVO, premuto una seconda volta ANNULLA la situazione di ATTIVO degli SHIFT.



Il tasto RETURN (da noi abitualmente indicato con CR) quando premuto da solo serve per far accettare al calcolatore una linea di programma, un comando o una serie di dati di INPUT, se premuto insieme a uno SHIFT muove il cursore alla prossima linea logica. Esso non genera mai un carattere di controllo visibile.

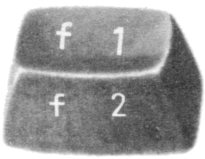


Il tasto CBM, come abbiamo già visto, permette di modificare il colore del cursore. Il SISTEMA OPERATIVO è in grado di riconoscere ogni volta che il tasto CBM viene premuto.
Se premi contemporaneamente SHIFT e CBM puoi osservare che sullo schermo avvengono alcuni cambiamenti:

- le lettere MAIUSCOLE diventano MINUSCOLE,
- i simboli GRAFICI di DESTRA dei SOLI tasti alfabetici si trasformano nelle lettere MAIUSCOLE riprodotte sui tasti a cui appartengono,
- gli altri simboli restano invariati.

Ciò succede perchè il tuo COMMODORE 64 dispone di due SET (insiemi o gruppi) di caratteri, normalmente indicati come SET MAIUSCOLO/GRAFICO (o STANDARD CBM) e SET MAIUSCOLO/MINUSCOLO (o COMMERCIALE), che possono essere utilizzati ALTERNATIVAMENTE, cioè a scelta (mai contemporaneamente), a seconda dell'uso che si vuole fare del calcolatore.
Il passaggio da un SET di caratteri all'altro avviene ogni volta che premi SHIFT-CBM.

TASTI FUNZIONE
Sono i 4 tasti disposti sulla destra della tastiera.



In un calcolatore nella sua configurazione base questi tasti non svolgono alcuna funzione particolare, ma poichè essi sono riconosciuti dal SISTEMA OPERATIVO possono essere utilizzati durante la selezione tra diverse possibilità in un programma grazie all'uso, ad esempio, dell'istruzione BASIC GET.
Inoltre, tramite il linguaggio macchina, è possibile assegnare ai diversi tasti funzione un significato particolare.
Su ognuno dei tasti compaiono due scritte: per generare la funzione scritta sopra (funzione dispari) devi premere il tasto da solo, mentre per generare quella scritta frontalmente (funzione pari) devi premere il tasto insieme a uno SHIFT.

Ti suggeriamo di provare a utilizzare tutti i tasti e verificare che il comportamento del calcolatore è quello previsto. Ti accorgerai di diverse cose:

- se premi CR mentre il cursore è su una linea che contiene caratteri messi a casaccio ottieni probabilmente la scritta SYNTAX ERROR e poi READY seguito dal punto; questo non succede se premi SHIFT-CR;
- se cerchi di far scendere il cursore più giù dell'ultima linea dello schermo, i contenuti di questo si spostano verso l'alto, liberando una o due linee in basso, ma perdendo i contenuti delle linee iniziali (effetto chiamato SCROLLING);
- dopo la pressione di alcuni tasti (SHIFT-INST/DEL e SHIFT-2) i tasti di controllo e comando non generano più un effetto immediato sullo schermo, ma fanno comparire dei caratteri in campo inverso.

Perchè succedono queste cose ti sarà chiaro andando avanti nella lettura.

2.2 L'INTERFACCIA CALCOLATORE-UTENTE

2.2.1 Screen Editor

Quando compare il messaggio iniziale di partenza del COMMODORE 64 è attiva quella parte di SISTEMA OPERATIVO che si chiama SCREEN EDITOR e che ti permette di colloquiare semplicemente con il tuo calcolatore, muovendoti come vuoi sullo schermo spostando il cursore per mezzo dei tasti di controllo. L'EDITOR, ogni volta che trovandoti in stato comandi premi il tasto CR, passa il contenuto della LINEA LOGICA sulla quale si trova il cursore all'INTERPRETE BASIC. Quest'ultimo controlla se la linea inizia con un numero, nel quale caso la memorizza come linea di programma nell'area di memoria riservata al programma BASIC; se la linea non inizia con un numero il sistema tenta di eseguirla e, se non ci riesce, segnala gli errori riscontrati. Di tutto questo si tratta diffusamente del volume dedicato al BASIC.
Vediamo ora cosa succede quando tu premi un tasto.

2.2.2 Buffer della tastiera (Keyboard Queue)

Ogni sessantesimo di secondo la tastiera viene scandita per controllare quale o quali tasti sono stati premuti (lavoro svolto dalle routine KERNAL del SISTEMA OPERATIVO) e il codice, generato dal tasto o dalla combinazione dei tasti, viene posto nel byte 197. Se il SISTEMA OPERATIVO non ne ha bisogno immediatamente, tenta di inserirlo, dopo averlo trasformato in ASCII, nella prima locazione libera di una serie di 10 locazioni contigue che inizia al byte 631 e termina al byte 640, detta BUFFER DELLA TASTIERA. Il buffer della tastiera viene gestito come una coda, trattando i caratteri nello stesso ordine nel quale essi vengono inseriti. Per gestire la coda viene usato un PUNTATORE, che risulta il CONTATORE del numero dei caratteri presenti nel buffer, esso è nel byte 198.

2.2.3 Modo tra virgolette (Quote Mode)

Se durante l'introduzione di una linea di comandi o di programma premi SHIFT-2 generi il carattere VIRGOLETTE (APICI) e entri in QUOTE MODE. In questa situazione la pressione dei tasti:
CRSR, CLR/HOME, CTRL-tasto numerico, CTRL-tasto alfabetico (escluso M), CBM-tasto numerico, da F1 a F8, non provoca un effetto immediato, ma la visualizzazione di un carattere in campo inverso: il carattere di controllo corrispondente.
Il QUOTE MODE perdura fino a quando fai una delle cose seguenti:

- premi una volta SHIFT-2, cioè generi un altro carattere VIRGOLETTE: in

effetti il sistema aggiunge 1, ogni volta che generi il carattere virgolette, nella locazione 212; è attivo lo stato QUOTE MODE se il numero di virgolette generato è dispari;

- premi CR o SHIFT-CR; questo chiude la linea e cancella il contenuto della locazione 212;
- premi RUN/STOP-RESTORE: drasticamente questo interrompe la fase di ingresso della linea di comando o programma, pulisce lo schermo, ripristina alcuni puntatori ai loro valori iniziali e ritorna allo stato di EDITOR.

2.2.4 Modo inserimento (Insert Mode)

Se premi SHIFT-INST/DEL mentre stai lavorando in EDITOR, il testo della linea logica a seguito del cursore è spostato a destra di tanti spazi quante sono le pressioni di INST, ammesso che ci sia spazio sulla linea. Questo stesso numero è memorizzato nel byte 216. Puoi allora introdurre altri caratteri, ma devi fare attenzione perchè la pressione dei tasti di comando/servizio e funzione provoca lo stesso effetto di quando sei in QUOTE MODE, con l'eccezione di DEL che produce una T in campo inverso e INST, che, invece, non fa altro che continuare a inserire spazi. Il MODO INSERIMENTO termina con la pressione di CR o con il riempimento di tutti gli spazi creati nel modo stesso.
La Tabella 2.2 che segue mostra l'aspetto dei caratteri di controllo relativi ai tasti di comando/servizio e funzione in QUOTE MODE e in INSERT MODE.

CARATTERI DI CONTROLLO IN QUOTE MODE

TASTO	COME APPARE
CRSR SU	"]"
CRSR GIU'	"N"
CRSR SINISTRA	"[illegible]"
CRSR DESTRA	"[illegible]"
CLR/HOME	"]"
HOME	"[illegible]"
INST/DEL	"[illegible]"
F1	"[illegible]"
F2	"[illegible]"
F3	"[illegible]"
F4	"[illegible]"
F5	"[illegible]"
F6	"[illegible]"
F7	"[illegible]"
F8	"[illegible]"

Tabella 2.2 Caratteri di controllo in QUOTE MODE

Ricordati che la presenza di questi caratteri di controllo in una istruzione di assegnazione (A$ =...) o stampa (PRINT...) produce poi gli stessi effetti che se venissero usati in modo immediato, cioè:

```
PRINT"[illegible]"
```

produce in stampa: CIAO seguito da READY, così come:

```
PRINT"I[illegible]"
```

genera sul video la scritta:

```
I
  N
    G
      I
        U
          ,
```

e:

```
PRINT"[illegible]"
```

produce la scritta: COLORI, con ogni lettera di un colore diverso, seguita da READY., scritto in verde (con il cursore lampeggiante nello stesso colore).

2.2.5 Il codice ASCII-CBM

Il COMMODORE 64 memorizza i dati di carattere alfanumerico in una forma particolare, utilizzando il codice ASCII-CBM, una delle tante versioni del codice ASCII.

Come puoi notare, nel set maiuscolo/minuscolo al codice 193 corrisponde il carattere A, mentre nel set maiuscolo/grafico al codice 193 corrisponde il carattere grafico a forma di picche. Puoi anche notare che i caratteri corrispondenti ai codici da 96 a 127 sono gli stessi di quelli corrispondenti ai codici da 192 a 223, ma il SISTEMA OPERATIVO li riconosce diversi.

I caratteri stampabili sono 128 più 128 in campo inverso e si ottengono con la pressione dei vari tasti della tastiera in combinazione anche con SHIFT e CBM.

2.2.6 Organizzazione dello schermo e Editor

Per poterti permettere l'interazione che abbiamo visto l'EDITOR deve essere in grado di rilevare cosa è presente sullo schermo in ogni istante e/o modificarlo, perciò deve memorizzare la posizione del cursore sullo schermo, il colore del cursore, il modo (diretto/inverso, quote, inserimento) e altri dati ancora.
Lo schermo è organizzato in una matrice di 25 linee di 40 caratteri ciascuna, rappresentata in memoria da due serie di 1000 locazioni, in ciascuna delle quali le prime 40 locazioni rappresentano la prima linea, le successive 40 la seconda linea e così via.
Le due aree di memoria da 1000 byte ciascuna sono:

1) MAPPA VIDEO: normalmente inizia all'indirizzo 1024, ma questo indirizzo può essere cambiato, come descritto nel seguito. In essa sono memorizzati i caratteri da visualizzare, utilizzando un codice diverso dal codice ASCII e chiamato DISPLAY CODE (D-CODE).

2) MAPPA COLORE: inizia sempre alla locazione 55296 e termina alla locazione 56295. Non è composta da 1000 byte, ma da 1000 NIBBLE, cioè semibyte, gruppi di 4 bit nei quali sono rappresentabili solo 16 valori diversi.

Le due aree di memoria si possono schematizzare come indicato in Figura 2.2.

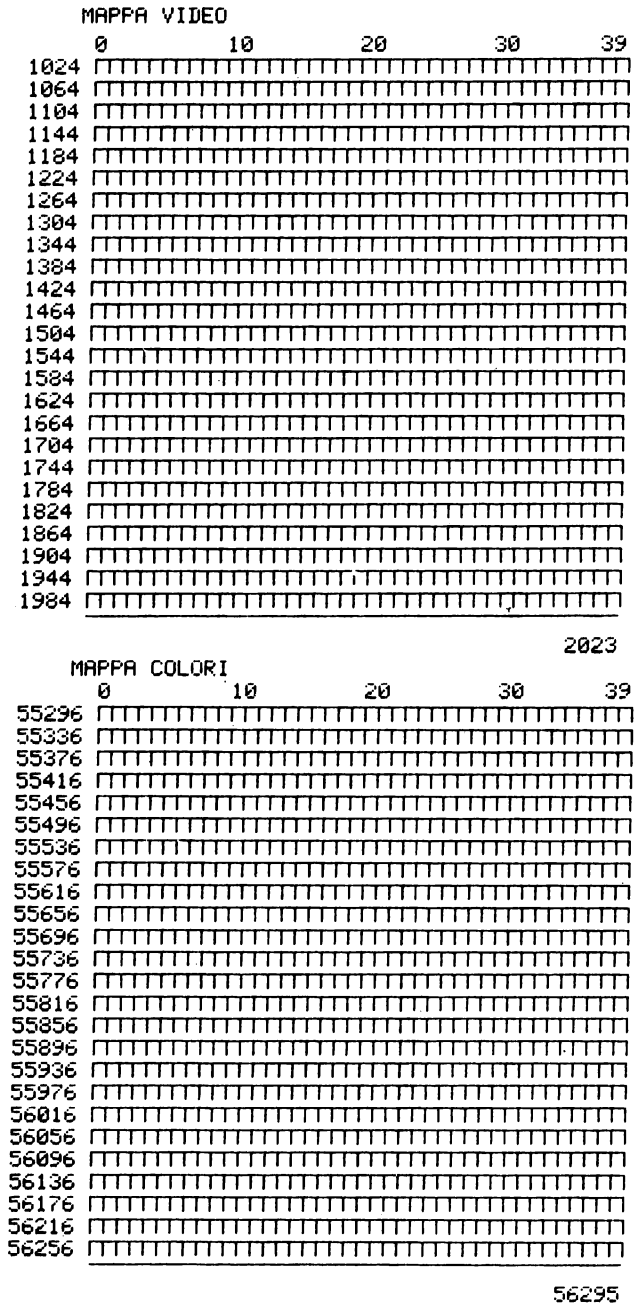


Figura 2.2 MAPPA VIDEO e MAPPA COLORI

Puoi accedere a ognuna di queste locazioni di memoria, sia per scrivere che per leggere; l'indirizzo di ognuno dei caratteri sullo schermo lo puoi calcolare sommando all'indirizzo del primo carattere della linea la posizione (0-39) del carattere sulla linea stessa.
Per far comparire un carattere sullo schermo occorre:

1) memorizzare nella MAPPA VIDEO il codice del carattere da visualizzare in D-CODE,

2) memorizzare nella locazione corrispondente della MAPPA COLORE un numero compreso tra 0 e 15, come numero del colore desiderato per il carattere.

Questo è esattamente quello che fa l'EDITOR, con la differenza che anzichè lavorare con le LINEE FISICHE (25 di 40 caratteri ciascuna), esso tratta LINEE LOGICHE, che possono essere lunghe fino a 80 caratteri.
Quando scrivi qualcosa sullo schermo l'EDITOR mantiene traccia dei movimenti del cursore in modo da sapere sempre:

- su quale linea logica dello schermo si trova il cursore, nel byte 214,
- in quale posizione (0-79) sulla linea logica si trova il cursore, nel byte 211,
- quale è l'indirizzo del primo carattere della linea logica del cursore nella mappa video ((byte 210)*256+(byte 209)) e nella mappa colore ((byte 244)*256+(byte 243)).

Il sistema può fare questo mantenendo una TABELLA DEI COLLEGAMENTI TRA LE LINEE (byte 217-242) di 26 byte, ognuno dei quali contiene un numero che indica se tale linea è o non è la prosecuzione della linea precedente. Tale numero è il risultato intero della divisione per 256 dell'indirizzo del primo carattere della linea fisica + 128 se la linea NON E' di continuazione, cioè il byte alto dell'indirizzo del primo carattere sulla linea fisica, eventualmente incrementato di 128. Questa tabella è aggiornata dopo ogni operazione di modifica dei contenuti dello schermo che interessi più di una linea fisica. Ad esempio, quando stai modificando una linea in un listato di programma, più corta di 40 caratteri, nel momento in cui tenti di scrivere il 40-esimo carattere il sistema si accorge che la linea fisica seguente non è parte della linea logica sulla quale stai scrivendo; crea allora sullo schermo una linea bianca, sulla quale tu potrai proseguire a scrivere, mentre la tabella dei collegamenti viene aggiornata.
Per quanto riguarda il calcolo dell'indirizzo del primo carattere di ogni linea il SISTEMA OPERATIVO si avvale di una tabella in ROM che contiene gli indirizzi di inizio delle 25 linee fisiche sullo schermo, quando la mappa video inizia in 1024, e da questa parte per effettuare i calcoli e per trovare tutti gli altri indirizzi e valori necessari.
Al momento in cui premi CR, il contenuto della linea logica sulla quale si trova il cursore (e che le vale la denominazione di LINEA CORRENTE) viene trascritto in una ZONA di memoria chiamata BUFFER DI INPUT, poichè è utilizzata per tutte le operazioni di ingresso dati. Essa inizia alla locazione 512 ed è lunga 89 byte; dato che dalla tastiera vengono letti solo 80 caratteri, in questo caso l'utilizzo non è completo.

L'EDITOR copia INTEGRALMENTE ciò che trova sul video e questo ti permette, ad esempio, tutte le correzioni e modifiche esaminate prima, e in più, di includere nelle stringhe di stampa caratteri di controllo, che normalmente non sono ottenibili da tastiera. Puoi operare così:

- lasciare lo spazio per i caratteri da aggiungere,
- porre il cursore in REVERSE MODE (CTRL-RVS ON),
- posizionarti dove vuoi inserire i caratteri di controllo,
- premere:
  - SHIFT-M, che appare come ▚ e corrisponde a SHIFT-CR,
  - N, che appare come ▞ e corrisponde a SHIFT-CBM nel set maiuscolo/grafico,
  - SHIFT-N, che appare come ▟ e corrisponde a SHIFT-CBM nel set maiuscolo/minuscolo,
  - H, che appare come ▗ e corrisponde alla disattivazione della combinazione SHIFT-CBM,
  - I, che appare come ▖ e corrisponde all'abilitazione della combinazione SHIFT-CBM.

Questi codici sono validi nel set ASCII-CBM e quindi il COMMODORE 64 sa cosa memorizzare.
ATTENZIONE però, dato che SHIFT-CR ha effetto ANCHE NEI LISTATI, esso può rendere difficile, se non impossibile, una correzione della linea che lo contiene.

Quando, muovendoti con il cursore o introducendo una linea, tenti di superare l'ultima linea dello schermo, il SISTEMA OPERATIVO ti procura uno spazio libero, spostando i contenuti dello schermo in su di una linea, se, invece, a eccedere la capacità dello schermo è un LISTATO, allora lo spostamento delle linee, detto scrolling, è di una linea logica, cioè di una o due linee fisiche.

2.2.7 Riferimenti in pagina zero

Mentre svolge le sue attività il SISTEMA OPERATIVO memorizza i dati che gli servono in alcune locazioni riservate, in particolare in quelle locazioni che hanno indirizzo compreso tra 0 e 255 (pagina 0), alle quali il microprocessore del COMMODORE 64 può accedere molto velocemente. Alcune di queste locazioni le abbiamo già viste, ma ce ne sono altre che riguardano la gestione del video e della tastiera.

- 9 — indica in quale posizione dello schermo si deve cominciare a scrivere dopo un TAB;

- 145 — indica la pressione del tasto STOP e di alcuni tasti, come indicato nella Tabella 2.3.

BYTE 145

STOP	Q	CBM	SPAZIO	2	CTRL	←	1

OGNI BIT CORRISPONDE A UN TASTO
IL BIT E' 0 PER TASTO PREMUTO
IN ASSENZA DI TASTI PREMUTI CONTIENE 255

Tabella 2.3 Contenuto BYTE 145

- 199 — <> 0 per RVS ON,
  > 0 per RVS OFF;

- 201-202 — al momento di eseguire una INPUT in questi due byte sono memorizzate le coordinate di schermo alle quali fare comparire il cursore;

- 646 — colore del cursore; modificare il valore di questo byte equivale ad impostare da tastiera o da programma un codice di controllo del cursore;

- 650 — indicazione di ripetizione automatica dei tasti:
  0 — per ripetizione solo dei 2 tasti CRSR, del tasto INST-/DEL e dello spazio (valore di default),
  1-127 — per disabilitare la ripetizione,
  128-255 — per ripetizione abilitata su tutti i tasti;

- 653 — tasto ausiliario premuto:
  bit di posizione 0 per SHIFT,
  bit di posizione 1 per CBM,
  bit di posizione 2 per CTRL,
  se i tasti sono premuti contemporaneamente vanno a 1 tutti i bit corrispondenti;

- 657 — 128 per disabilitare SHIFT-CBM,
  0 per abilitare SHIFT-CBM.

2.3 IL VIDEO

Nel tuo COMMODORE 64 c'è un circuito integrato chiamato VIC II (Video Interface Chip II) che tra le sue funzioni ha quella di attingere le informazioni contenute in certe regioni di memoria e utilizzarle per creare dei segnali che modulati e amplificati raggiungono il televisore o il monitor e creano un'immagine, tipicamente il riquadro per i caratteri circondato dal bordo colorato. Il VIC II è un dispositivo programmabile; questo significa che può funzionare in più di un modo, a seconda dei comandi che gli giungono in ingresso. In particolare, il VIC II può utilizzare le informazioni che legge dalla memoria in molti modi diversi; in questo manuale li vedrai utilizzati tutti.

MODO CARATTERI STANDARD
Quando il VIC II è programmato per utilizzare i dati in memoria per generare un quadro video di 25 linee da 40 caratteri ciascuna, in cui ogni carattere può essere di uno tra i 16 colori possibili, e condivide con gli altri 999 caratteri lo stesso sfondo (anche esso di uno tra i 16 colori possibili), contornato da un bordo colorato (esso pure di uno tra i 16 colori possibili), cioè si trova nella situazione disponibile al momento dell'accensione, diciamo che il COMMODORE 64 si trova nel MODO CARATTERI STANDARD.

2.3.1 Display Code

In modo caratteri, ognuno dei 1000 caratteri visualizzabili sullo schermo può essere scelto tra i 128 stampabili in campo diretto e i 128 stampabili in campo inverso, cioè tra 256 caratteri diversi, ognuno dei quali può essere rappresentato da un codice in un byte della mappa video.
Il codice usato per individuare ognuno dei 256 caratteri è ovviamente diverso dal codice ASCII-CBM, in quanto non comprende i caratteri di controllo, ed è chiamato DISPLAY CODE.
I DISPLAY CODE dei caratteri in campo diretto e inverso si trovano nelle due ultime colonne delle tabelline riportate nel secondo Volume. Naturalmente esiste una certa corrispondenza tra CODICE ASCII e DISPLAY CODE; le regole che permettono il passaggio da un codice all'altro sono riportate nella Tabella 2.4.

ASCII	D/CODE
32 <= A <= 63	D = A
64 <= A <= 95	D = A - 64
96 <= A <= 127	D = A - 32
160 <= A <= 191	D = A - 64
192 <= A <= 254	D = A - 128
A = 255	D = A - 161

Tabella 2.4 Relazione tra CODICE ASCII e D-CODE

Per leggere o scrivere un carattere sullo schermo bisogna calcolare l'indirizzo della MAPPA VIDEO in cui eseguire una PEEK o una POKE. Se consideriamo le linee numerate da 0 a 24, dall'alto in basso, e le posizioni dei caratteri da 0 a 39, da sinistra a destra, l'indirizzo si calcola come:

ind.MV=40*num.linea+posiz.car.+ind.base MV
All'accensione l'indirizzo base della mappa video è 1024.

2.3.2 Colore caratteri, sfondo e bordo

In modo caratteri il colore di ognuno dei caratteri può essere scelto tra i 16 disponibili senza alcuna limitazione. Nella mappa colore sono disponibili 4 bit per il colore di ogni carattere e questi bastano per rappresentare un numero tra 0 e 15. La corrispondenza tra codici colore e colori è la seguente:

0..nero	10...rosso-chiaro
1..bianco	11...grigio-1
2..rosso	12...grigio-2
3..ciano	13...verde-chiaro
4..porpora	14...azzurro
5..verde	15...grigio-3
6..blu	
7..giallo	
8..arancione	
9..marrone	

L'indirizzo in cui eseguire la lettura (PEEK) o la scrittura (POKE) del codice colore si ricava in modo analogo a quello per la mappa video:

ind.MC=55296+40*num.linea+pos.carattere

Quando effettui una lettura devi ricordarti di considerare solo i 4 bit meno significativi del valore ottenuto, ovvero, se A è l'indirizzo della mappa colore di un certo carattere (poichè i 4 bit più significativi, che corrispondono ad altrettante linee non connesse, possono essere valori casuali):

codice colore= PEEK(A) AND 15

Lo stesso sistema di codifica dei colori è usato per lo sfondo comune a tutti i caratteri (detto BACKGROUND COLOR), e per il colore del bordo (detto BORDER COLOR). Per modificare questi due colori bisogna accedere a due diverse locazioni di memoria:

per lo sfondo: per scrivere: POKE 53281, colore
per leggere: C=PEEK(53281) AND 15

per il bordo: per scrivere: POKE 53280, colore
per leggere: C=PEEK(53280) AND 15

L'operazione AND è necessaria poichè anche i primi 4 bit delle due locazioni lette corrispondono a 4 linee non connesse, e vanno quindi mascherati.

2.3.3 Visualizzazione

Il D-CODE di ogni carattere viene utilizzato dai circuiti di controllo del VIC II per ritrovare in memoria la descrizione del carattere stesso per punti. Ognuno dei 512 caratteri visualizzabili sullo schermo (256 per ognuno dei due set disponibili) è descritto in memoria da una sequenza di 8 byte. Nel modo caratteri standard ogni bit 1 corrisponde a un punto nel colore scelto per il carattere, ogni bit 0 corrisponde a un punto nel colore scelto per lo sfondo.
Il D-CODE viene usato come indice per calcolare l'indirizzo di inizio della sequenza di 8 byte (cioè l'indirizzo del primo) nel seguente modo:

ind.=ind.base descr.caratt.+8*D-CODE

L'indirizzo base della descrizione dei caratteri, che chiamiamo BC da Base Character, può essere modificato a seconda della necessità dell'utente. Nel modo caratteri standard:

ind.=53248+8*D-CODE nel set maiuscolo/grafico
ind.=55296+8*D-CODE nel set maiuscolo/minuscolo.

Ad esempio, il carattere "&" viene rappresentato nei byte da 53552 a 55559 nel set maiuscolo/grafico, e nei byte da 55600 a 55607 nel set maiuscolo/minuscolo, come segue (il primo numero è la rappresentazione binaria, il secondo quella decimale), e appare come indicato a lato:

```
00000000 ..................... 0 . . . . . . . .
00111100 ..................... 60 . . * * * * . .
01100110 .................... 102 . * * . . * * .
00111100 ..................... 60 . . * * * * . .
00111000 ..................... 56 . . * * * . . .
01100111 .................... 103 . * * . . * * *
01100110 .................... 102 . * * . . * * .
00111111 ..................... 63 . . * * * * * *
```

e il carattere "@" nei byte da 53248 a 53255 e nei byte da 55296 a 55303, così:

```
00000000 ..................... 0 . . . . . . . .
00111100 ..................... 60 . . * * * * . .
01000110 ..................... 70 . * * . . * * .
01001110 ..................... 78 . * * . * * * .
01001110 ..................... 78 . * * . * * * .
01000000 ..................... 64 . * * . . . . .
01000010 ..................... 66 . * * . . * * .
00111100 ..................... 60 . . * * * * . .
```

Se hai letto con attenzione a questo punto penserai: MA QUI C'E' UN ERRORE! Infatti la descrizione del carattere "@" comincia nel set maiuscolo/minuscolo esattamente nella stessa locazione dove inizia la MAPPA COLORE. Come può essere? La spiegazione è che nel COMMODORE 64, a partire dall'indirizzo 53248, coesistono un BANCO di 4K RAM, uno di 4K I/O ed uno di 4K ROM, contenente le descrizioni dei caratteri. Normalmente l'accesso è permesso ai 4K I/O, utilizzati per le operazioni di ingresso e uscita, e le descrizioni dei caratteri in ROM sono inacessibili (prova ad effettuare qualche PEEK di controllo). Quando il VIC II ha bisogno delle descrizioni dei caratteri per generare l'immagine sul video, effettua una operazione detta di "bank switching" (scambio di banchi) che in pratica SOSTITUISCE logicamente il banco di memoria ROM al banco di memoria RAM per il tempo necessario.
Esiste comunque un modo per effettuare lo "switching" da BASIC. Per farlo devi procedere così:

- disabilitare le interruzioni, cioè le chiamate di servizio da parte di altri dispositivi esterni; questo praticamente disabilita la tastiera;
- disabilitare l'ingresso e l'uscita e sostituire il banco RAM con il banco ROM; così risultano bloccate tutte le operazioni di ingresso e uscita.

Per ottenere questo in BASIC, devi scrivere:

1) POKE 56334,PEEK(56334) AND 254

2) POKE 1, PEEK(1) AND 251

a questo punto risulta accessibile la ROM agli indirizzi 53248-57343 e puoi leggere le descrizioni dei caratteri; per riportare il sistema nelle condizioni normali devi eseguire:

3) POKE 1,PEEK(1) OR 4

4) POKE 56334,PEEK(56334) OR 1.

Il programma ES0 esemplifica il prelevamento dalla memoria dei caratteri dei due set.

```
 1 REM ES0
 3 REM MOSTRA CARATTERI
 5 BO=53280:SF=BO+1:CO=646
 10 DIMA%(7):POKEBO,14:POKESF,6:POKECO,15
 15 OPEN7,3
 20 INPUT"RISULTATO SU STAMPANTE (S/N)";R$
 25 AS=1+(R$="N")+2*(R$="S"):IFAS=1THEN20
 30 INPUT"SET (C)OMMERCIALE O (G)RAFICO";SET$
 33 IFSET$<>"C"ANDSET$<>"G"THEN30
 35 BC=53248-2048*(SET$="C")
 40 INPUT"D - CODE (0 - 255)";D
 45 IFD<0ORD>255THEN40
 50 PRINT"";:POKE1024,D:GET#7,A$:A=ASC(A$)
 55 POKE212,0:PRINT"";
 60 POKE56334,PEEK(56334)AND254
 65 POKE1,PEEK(1)AND251
 70 B=BC+8*D:FORK=0TO7:A%(K)=PEEK(B+K):NEXT
 80 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
 90 GOSUB300
 100 IFAS=-1THEN150
 110 INPUT"ANCORA (S/N)";R$:IFR$="S"THEN20
 120 IFR$<>"N"THEN110
 125 CLOSE7
 130 PRINT"":END
 150 OPEN4,4,0:CMD4:GOSUB300:PRINT#4:CLOSE4
 155 GOTO110
 300 RVS=-(D>127)
 305 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ D CODE =";
```

```
306 PRINTRIGHT$("  "+STR$(D),4)
310 FORK=0TO7:PRINT"▒";
315 FORJ=0TO7
316 PRINTCHR$(32-81*((2↑(7-J) AND A%(K))<>0));
317 NEXT J
320 PRINT"▒";:IFK<>2THEN330
325 PRINT" ASCII="RIGHT$("  "+STR$(A),4);
326 PRINT" (RVS "MID$("OFFON",1+3*RVS,3-RVS)")";
330 PRINT:NEXT
335 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
340 RETURN
```

2.4 LAVORARE IN MODO CARATTERI

Anche senza avere a disposizione della grafica sofisticata è possibile scrivere dei programmi utili e interessanti, dai programmi gestionali ai programmi di trattamento testi, a programmi matematici o educativi.
Tutte le volte che è sufficiente un RESPONSO di natura alfanumerica da parte del programma può essere utilizzato il modo caratteri. Riguardo a ciò ci sono almeno tre argomenti da trattare:

1) input controllato,
2) maschere video,
3) presentazione dei dati.

Di questi argomenti abbiamo già parlato nel volume sul BASIC, ma ci sembra utile ritornare a trattarne anche qui, per completezza.

2.4.1 Input controllato

Con questo termine si intende l'introduzione controllata dei dati e questa può essere ottenuta:

A) controllando la validità del dato in ingresso dopo averlo letto;
B) controllando la validità del dato in ingresso durante la sua introduzione.

Il controllo A) può essere fatto dopo una istruzione INPUT, una READ o una INPUT #; normalmente consiste nel controllare che un dato ricevuto soddisfi a determinate condizioni (da notare che, se alcune condizioni non sono verificate, l'errore viene segnalato dal sistema):

- non sia più corto o più lungo di una lunghezzza fissata,
- non contenga caratteri indesiderati, come caratteri di controllo,
- sia del tipo desiderato, cioè contenga solo caratteri alfabetici o solo caratteri numerici (dopo averlo letto come stringa),
- sia limitato in valore, cioè un numero compreso in un intervallo, o un dato alfabetico che rispetti un certo ordinamento,

- soddisfi altri vincoli inerenti all'argomento trattato.

In caso contrario si può agire in diversi modi:

- si segnala l'errore e si richiede una nuova lettura,
- si segnala l'errore e si conclude la procedura in atto,
- si corregge il dato in ingresso e se ne chiede conferma.

Vediamo come è possibile realizzare i diversi controlli.

LUNGHEZZA

Il controllo è effettuato con la funzione LEN(A$), che fornisce la lunghezza della stringa data, come mostrato nel programma ES1.

```
1 REM ES1
110 INPUT"▯▯▯NOME: ";A$
115 IFLEN(A$)=0ORLEN(A$)>20THEN110
```

Considerando le caratteristiche dell'EDITOR di schermo è conveniente talvolta cancellare prima il vecchio dato, come indicato nel programma ES2.

```
1 REM ES2
5 S$="":FORI=1TO80:S$=S$+" ":NEXTI
110 PRINT"▯▯▯"S$"▯▯";
113 INPUT"NOME: ";A$
115 IFLEN(A$)=0ORLEN(A$)>20THEN110
```

Il controllo sulla lunghezza del dato può essere eliminato se desiderato, ad esempio quando si include nella stringa di INPUT una serie di caratteri che segnalano la lunghezza desiderata, come mostrato nel programma ES3.

```
1 REM ES3
110 PRINT"▯▯▯";
113 INPUT"NOME: ------▮▮▮▮▮▮▮▮▮";A$
```

PRESENZA CARATTERI INDESIDERATI

Possono essere controllati singolarmente i caratteri della stringa ricevuta, eliminando quelli indesiderati o segnalando errore, come in questo caso, dove il dato in ingresso è in A$, e in uscita si pone E=0 se la stringa è corretta ed E=1 se non lo è, come mostrato nel programma ES4.

```
1 REM ES4
5 REM PROVA ROUTINE CONTROLLO
10 PRINT"□STRINGA DA CONTROLLARE"
20 INPUTA$:GOSUB100
30 PRINT"STRINGA "MID$("NON CORRETTA",5-4*E)
40 INPUT"□□PREMI [RETURN]";I$
50 RUN
100 REM ROUTINE CONTROLLO
105 FORK=1TOLEN(A$):A1$=MID$(A$,K,1)
110 E=-(A1$<" "ORA1$>"]"ANDA1$<"♠"ORA1$>"♦")
115 IFETHENRETURN
120 NEXT:RETURN
```

Alla linea 110 viene assegnato ad E un valore che è 1 se l'espressione logica tra parentesi (che effettua il controllo sulla non validità del carattere puntato dall'indice K in A$) è vera, 0 se essa è falsa.
Se E=1 il ciclo viene concluso. E' possibile segnalare la posizione del carattere errato memorizzandola in una variabile P, come indicato nel programma ES4.1.

```
1 REM ES4.1
5 REM PROVA ROUTINE CONTROLLO
10 PRINT"□STRINGA DA CONTROLLARE"
20 INPUTA$:GOSUB100
25 IFETHENPRINTSPC(1+P)"↑"
30 PRINT"STRINGA "MID$("NON CORRETTA",5-4*E)
40 INPUT"□□PREMI [RETURN]";I$
50 RUN
100 REM ROUTINE CONTROLLO
105 FORK=1TOLEN(A$):A1$=MID$(A$,K,1)
110 E=-(A1$<" "ORA1$>"]"ANDA1$<"♠"ORA1$>"♦")
115 IFETHENP=K:RETURN
120 NEXT:RETURN
```

E' anche possibile, ogni volta che si incontra un carattere non valido, eliminarlo o sostituirlo con un determinato carattere, ad esempio, lo spazio. Ad esempio: eliminare i caratteri non alfanumerici o di interpunzione dalla stringa A$. Il dato di ingresso è in A$, il dato in uscita eventualmente corretto è in A$; E=1 se A$ è stata modificata, E=0 se no. Vedi al riguardo come abbiamo realizzato il programma ES5.

```
1 REM ES5
5 REM PROVA ROUTINE CONTROLLO
10 PRINT"▒STRINGA DA CONTROLLARE"
20 INPUTA$:GOSUB100
30 PRINT"STRINGA "MID$("NON CORRETTA",5-4*E1)
35 IFE1THENPRINT"▒STRINGA CORRETTA:":PRINTA$
40 INPUT"▒▒PREMI [RETURN]";I$
50 RUN
100 REM ROUTINE CONTROLLO
105 E1=0:K1=0
107 FORK=1TOLEN(A$):K1=K1+1:A1$=MID$(A$,K1,1)
110 E=-(A1$<" "ORA1$>"]"ANDA1$<"↑"ORA1$>"←")
115 IF E THEN125
120 NEXT:RETURN
125 A$=LEFT$(A$,K1-1)+MID$(A$,K1+1)
126 E1=1:K1=K1-1:GOTO120
```

Si può modificare la linea 115 in modo che al posto del carattere non valido venga inserito il carattere "?", come nel programma ES5.1.

```
1 REM ES5.1
5 REM PROVA ROUTINE CONTROLLO
10 PRINT"▒STRINGA DA CONTROLLARE"
20 INPUTA$:GOSUB100
30 PRINT"STRINGA "MID$("NON CORRETTA",5-4*E1)
35 IFE1THENPRINT"▒STRINGA CORRETTA:":PRINTA$
40 INPUT"▒▒PREMI [RETURN]";I$
50 RUN
100 REM ROUTINE CONTROLLO
105 E1=0:K1=0
107 FORK=1TOLEN(A$):K1=K1+1:A1$=MID$(A$,K1,1)
110 E=-(A1$<" "ORA1$>"]"ANDA1$<"↑"ORA1$>"←")
115 IF E THEN125
120 NEXT:RETURN
125 A$=LEFT$(A$,K1-1)+"?"+MID$(A$,K1+1)
126 E1=1:GOTO120
```

TIPO

Se si effettua la INPUT di un dato numerico e si tenta di introdurre un dato alfanumerico, il sistema segnala "?REDO FROM START" e richiede il dato. Poichè non è possibile eliminare questo inconveniente che spesso porta a confusione sullo schermo, scrolling o altro, è preferibile effettuare la lettura con una variabile stringa e controllare la correttezza del dato da programma. Ciò provoca un rallentamento nell'esecuzione, ma evita altri inconvenienti. Per esempio: controllare se A$ contiene un numero intero senza segno. Il dato in ingresso è in A$: in uscita E=0 se va bene, E=1 se no. Segue il programma ES6.

```
1 REM ES6
5 REM PROVA ROUTINE CONTROLLO
10 PRINT"STRINGA DA CONTROLLARE"
20 INPUTA$:GOSUB100
30 PRINT"STRINGA "MID$("NON NUMERICA",5-4*E)
40 INPUT"PREMI [RETURN]";I$
50 RUN
100 REM ROUTINE CONTROLLO
105 E=0:FORK=1TOLEN(A$):A1$=MID$(A$,K,1)
110 E=-(A1$<"0"ORA1$>"9")
115 IFETHENK=LEN(A$)
120 NEXT:RETURN
```

E' possibile aggiungere il controllo sulla presenza del segno modificando le linee 100 e 105, come indicato nel programma ES7.

```
1 REM ES7
5 REM PROVA ROUTINE CONTROLLO
10 PRINT"STRINGA DA CONTROLLARE"
20 INPUTA$:GOSUB100
30 PRINT"STRINGA "MID$("NON NUMERICA",5-4*E)
40 INPUT"PREMI [RETURN]";I$
50 RUN
100 REM ROUTINE CONTROLLO
101 E=0:A=1-(LEFT$(A$,1)="+"ORLEFT$(A$,1)="-")
105 FORK=ATOLEN(A$):A1$=MID$(A$,K,1)
110 E=-(A1$<"0"ORA1$>"9")
115 IFETHENK=LEN(A$)
120 NEXT:RETURN
```

In questo modo, se il primo carattere della stringa numerica è + o -, il controllo di validità parte dal secondo carattere. E' possibile introdurre anche un controllo sulla presenza di un solo punto decimale in testa al numero oppure all'interno, utilizzando una variabile PD. Abbiamo realizzato come esempio il programma ES8.

```
1 REM ES8
5 REM PROVA ROUTINE CONTROLLO
10 PRINT"STRINGA DA CONTROLLARE"
20 INPUTA$:GOSUB100
30 PRINT"STRINGA "MID$("NON NUMERICA",5-4*E)
40 INPUT"PREMI [RETURN]";I$
50 RUN
100 REM ROUTINE CONTROLLO
101 E=0:A=1-(LEFT$(A$,1)="+"ORLEFT$(A$,1)="-")
103 PD=0
105 FORK=ATOLEN(A$):A1$=MID$(A$,K,1)
110 E=-(A1$<"0"ORA1$>"9")
112 IFEAND(A1$=".")ANDPD=0THENPD=K:E=0
115 IFETHENK=LEN(A$)
120 NEXT
125 IFPD=LEN(A$)THENE=1
130 RETURN
```

Includere il controllo su un generico numero reale vorrebbe dire:

- cercare la presenza del carattere E,
- controllare che i caratteri seguenti rappresentino un intero con segno, compreso entro determinati limiti, ma ciò complica parecchio le cose.

VALORE

Questo controllo è fatto per accertarsi che il dato in ingresso sia compreso entro certi limiti. Per i dati numerici la cosa è abbastanza semplice:

```
200 IF VAL(A$) <"0" OR VAL(A$)>"24" THEN PRINT "NON VALIDO!" :
GOTO 100
```

se il dato è letto come stringa; oppure:

```
200 IF A < 10 OR A > 100 THEN...
```

quando il dato è letto come numero.
Per i dati stringa bisogna ricordare che per l'ordinamento viene considerato il codice ASCII-CBM di ogni carattere e il numero dei caratteri della stringa; così:

"AIUTO " > "AIUTO"
"13" > "11"
"barca" < "Barca"

e il confronto può essere effettuato come:

210 IF A$ < B$ THEN...
oppure: 210 IF A$ > "11" THEN...

Il controllo B) viene effettuato con gli stessi controlli sul carattere già visti negli esempi, ma leggendo i caratteri singolarmente con un'istruzione GET (o una PEEK dalle locazioni di indirizzo noto); in questo caso è possibile non accettare un carattere non valido e proseguire.
Un problema legato all'introduzione dei dati carattere per carattere è la visualizzazione dei dati introdotti (feedback). Infatti, a differenza di INPUT, GET non visualizza nè un cursore lampeggiante nè tantomeno il carattere introdotto (la cosa non è automatica, il video e la tastiera sono due periferiche indipendenti); bisogna perciò provvedere da programma.
Per rimediare alla mancanza di feedback è spesso sufficiente un'istruzione che stampi il carattere una volta che questo sia considerato valido. Fanno eccezione:

- il carattere virgolette, per il quale bisogna avere particolare attenzione, perchè fa entrare e uscire dal QUOTE MODE anche in fase di stampa,
- i caratteri di controllo che eventualmente devono essere riconosciuti ed eseguiti.

Il problema di un cursore che visualizzi la posizione dove dovrà essere inserito il prossimo carattere può essere risolto in vari modi:

- se non è permesso tornare indietro sulla stringa appena letta è spesso sufficiente un qualunque carattere, magari in REVERSE o di colore differente, che indichi dove verrà aggiunto il prossimo carattere;
- se è permesso modificare la stringa riscrivendoci sopra si possono utilizzare tecniche del tipo:
  - cursore fisso, in campo inverso, che indica il carattere sul quale si trova;
  - cursore lampeggiante che mostra alternativamente il carattere sul quale si trova e/o lo stesso carattere in campo inverso o un altro carattere prefissato (di solito lo spazio o una lineetta);
  - cursore sottostante la stringa in lettura, che indica quale carattere verrà modificato o aggiunto premendo un tasto. A seconda delle applicazioni possono essere studiati altri tipi di presentazione del cursore.

Inoltre può essere utile controllare il colore dello sfondo (background) e quindi attribuire al cursore, e all'input visualizzato, un colore contrastante con esso.

Programma esempio ES9: legge una stringa solo alfanumerica, di lunghezza massima fissata, trasforma il carattere virgolette nel carattere apice. DEL cancella l'ultimo carattere, SHIFT- CLR/HOME cancella tutta la stringa. X e Y sono le coordinate di schermo dove iniziare l'INPUT (Y < 24); N è la lunghezza massima (N < 40), A$ è la stringa letta.

```
1 REM ES9
4 REM INPUT CONTROLLATO
5 B$="▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
6 S$="                                        "
7 C$="▄▄▌"
10 N=20:X=0:Y=5:PRINT"□"LEFT$(B$,Y+2)SPC(X);
15 FORK=1TON:PRINT"+";:NEXT:GOSUB100
20 STOP
100 REM ROUTINE CONTROLLO
105 A$="":PRINTLEFT$(B$,Y+1)SPC(X)LEFT$(S$,N)
110 PRINTLEFT$(B$,Y+1)SPC(X)C$;
115 FORK=0TO1:K=0
120 GETH$:IFH$=""THEN120
125 IFH$=CHR$(13)THENPRINT" ▌";:K=1:NEXT:RETURN
130 IFH$=CHR$(20)ANDLEN(A$)THEN155
135 IFH$=CHR$(147)THENK=1:PRINT" ▌";:NEXT:GOTO105
140 IFH$=CHR$(34)THENH$="'"
145 IFLEN(A$)<NTHEN160
150 NEXT
155 A$=LEFT$(A$,LEN(A$)-1):PRINT" ▌▌"C$;:NEXT
156 GOTO135
160 IF(H$>=" "ANDH$<="["ORH$>"-"ANDH$<"+")THEN170
165 GOTO150
170 A$=A$+H$:PRINTH$C$;:GOTO150
```

Programma esempio ES10 : legge una stringa solo numerica, massimo N cifre totali con un punto decimale. Sono utilizzati i caratteri di controllo DEL, SHIFT-CLR/HOME e movimento cursore orizzontale.

```
1 REM ES10
4 REM INPUT CONTROLLATO
5 B$="▀▀▀
```

ES11, come potrebbe essere la routine se si utilizzasse un cursore posto sotto la stringa di ingresso (la riga dove effettuare l'input non deve essere l'ultima in basso, cioè Y < 24).

```
1 REM ES11
4 REM INPUT CONTROLLATO
5 B$="▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
6 S$="                                        "
8 C$="▐▌":Q$=" ▌"
10 X=9:Y=10:N=10
15 PRINT"▒":GOSUB200
20 PRINT:PRINT:PRINT"NUMERO LETTO:"A$:STOP
200 REM ROUTINE CONTROLLO
205 PD=0:W=0
208 A$="":C=1
210 PRINTLEFT$(B$,Y+1)SPC(X)LEFT$(S$,N+1):PD=0
215 PRINTLEFT$(B$,Y+1)SPC(X)A$" ";
218 IFPDTHENPD=PD+PD*(MID$(A$,PD,1)<>".")
220 PRINTLEFT$(B$,Y+2)SPC(X+C-1)C$;
225 GETH$:IFH$=""THEN225
230 IFH$=CHR$(13)ANDPD<LEN(A$)THENPRINTQ$;:RETURN
235 IFH$=CHR$(147)THENPRINTQ$;:GOTO205
240 IFC=LEN(A$)+1ANDLEN(A$)>=NTHEN260
245 IFH$="."AND(PD=0)THENPD=C:W=1
250 IFH$>"/"ANDH$<":"THENW=1
255 IFWTHEN290
260 IFH$=CHR$(29)THEN297
265 IFH$=CHR$(157)THENC=C+(C>1):PRINTQ$;:GOTO220
270 IFH$<>CHR$(20)THEN225
275 IFC<2ORLEN(A$)=0THEN215
280 PD=PD+PD*(PD+1=C)+(C<PD)
285 A$=LEFT$(A$,C-2)+MID$(A$,C):C=C-1:PRINTQ$;
286 GOTO215
290 W=0:A$=LEFT$(A$,C-1)+H$+MID$(A$,C+1)
295 C=C+1:PRINTQ$;:GOTO215
297 C=C-(C<LEN(A$)):PRINTQ$;:GOTO220
```

2.4.2 Maschere video

Questo termine indica l'uso di certe scritte fisse sul video, durante l'introduzione dei dati, che creano appunto una MASCHERA di guida alla scrittura dei dati. Il modo

più semplice per creare una maschera video è probabilmente quello di scrivere in colonna tutte le voci da richiedere; in fase di lettura ci si posizionerà dopo ognuna delle voci e si eseguirà la lettura.
Si possono complicare e rendere più estetiche le cose associando ad ognuna delle voci una posizione X,Y sullo schermo, alla quale effettuare la lettura del dato. Inoltre, se i dati in ingresso sono relativi a un argomento specifico è consigliabile ricordarlo a chi usa il programma con delle scritte appropriate. Rimandiamo per gli esempi ai programmi QUADRO1 e QUADRO2 del primo Volume.

2.4.3 Presentazione dei dati

Riprendiamo qui l'argomento della presentazione dei dati sul video in modo ordinato (si veda il secondo Volume, che tratta l'argomento anche per la stampante).
Non avendo ancora a disposizione la grafica, per i dati numerici possiamo accontentarci di tabulati, e per dati più strutturati di maschere video.
Grazie alle funzioni TAB(X), che posiziona il cursore alla X-esima posizione a partire dall'inizio della linea corrente, SPC(X), che spazia di X caratteri SENZA SOVRASCRIVERE, e POS(X), che fornisce la posizione del cursore sulla linea LOGICA corrente (0 <=X <=79), è possibile incolonnare come si desidera i dati. Vediamo qualche esempio.

TAVOLA PITAGORICA: è un esempio classico di incolonnamento di dati numerici, non tutti dello stesso numero di cifre. Il risultato è riportato nella Tabella 2.5.

TAVOLA PITAGORICA

	1	2	3	4	5	6	7	8	9	10
1		2	3	4	5	6	7	8	9	10
2		4	6	8	10	12	14	16	18	20
3		6	9	12	15	18	21	24	27	30
4		8	12	16	20	24	28	32	36	40
5		10	15	20	25	30	35	40	45	50
6		12	18	24	30	36	42	48	54	60
7		14	21	28	35	42	49	56	63	70
8		16	24	32	40	48	56	64	72	80
9		18	27	36	45	54	63	72	81	90
10		20	30	40	50	60	70	80	90	100

Tabella 2.5 Tavola Pitagorica ottenuta con ES12

L'unico problema è quello di spaziare 3 caratteri se il numero è minore di 10, 2 caratteri se esso è maggiore di 10 e minore di 100, 1 carattere se esso è maggiore di 99. Un'espressione che dà tali valori in corrispondenza dei controlli indicati (K è il numero) è:
(1 - (K < 10) - (K < 100)).
Devi ricordare che un numero intero viene stampato preceduto dal segno - o da uno spazio e seguito da uno spazio.
Un programma per realizzare la tavola pitagorica è ES12; esso utilizza la funzione SPC.

```
1 REM ES12
5 REM TAVOLA PITAGORICA
10 PRINT"▐  TAVOLA  PITAGORICA";
15 PRINTSPC(22)"------------------"
20 FORK=1TO10
25 PRINT"▌▌"SPC(1-(K<10));K;:NEXTK:PRINT
30 FORK=1TO10
35 PRINT"▌▌"SPC(-(K<10));K;SPC(1);
40 FORJ=2TO10
45 A=K*J:PRINT"▌▌"SPC(-(A<100)-(A<10));A;
50 NEXTJ:PRINT"▐";:NEXTK
60 PRINT"▐";
70 GOTO70
```

Per utilizzare la funzione TAB si devono fare delle modifiche al programma e fare attenzione al fatto che: la funzione TAB non agisce sul video se l'argomento definisce una posizione già superata dal cursore.

LANCIO DI DUE DADI: il programma ES14 visualizza i dati relativi ad una simulazione del lancio di due dadi non truccati (per truccarli, basta modificare la funzione definita alla linea 30), ovvero il numero delle estrazioni avvenute e previste di ognuna delle 11 possibili somme. Dopo ogni estrazione i dati vengono aggiornati.

```
1 REM ES14
5 REM LANCIO DADI
7 PRINT"▐";
10 INPUT"NUMERO LANCI (L%>0 & L%<=1000)";L%
15 IFL%<1ORL%>1000THEN10
25 DIMFO(10),FP(10)
```

```
30 DEFFND(X)=INT(RND(0)*6)+1
35 FORK=0TO10
36 FP(K)=(6-ABS(5-K))/36:NEXTK
40 FORJ=1TOL%:A=FND(0)+FND(0)-2:FO(A)=FO(A)+1
47 PRINT"□ SOMMA"SPC(10)"ESTRAZIONI"SPC(16);
48 PRINT"DADI"SPC(5)"OSSERVATE   PREVISTE":PRINT
50 FORK=0TO10
51 PRINTTAB(1-(K<8))K+2;
53 X=FO(K)
54 PRINTTAB(12-(X<1000)-(X<100)-(X<10));FO(K);
55 X=INT(FP(K)*J*100+.5)/100
57 PRINTTAB(22-(X<1000)-(X<100)-(X<10)-(X<1))X
60 NEXTK
65 PRINT"■N. LANCI: ";
70 PRINTSPC(2-(J<1000)-(J<100)-(J<10))J"/"L%;
75 FORX=1TO400:NEXTX:NEXTJ
```

TIPI DI INCOLONNAMENTO: può essere studiato un incolonnamento adatto ai tipi di dati da visualizzare. Nell'esempio che segue i dati sono letti da linee DATA nel programma, in particolare i numeri reali sono scelti in modo da non essere trasformati nella corrispondente forma esponenziale (per non complicare i controlli). Abbiamo realizzato il programma ES15 come esempio.

```
1 REM ES15
5 REM INCOLONNAMENTI
7 S$="00000":RESTORE
10 PRINT"□■ALFABETICO---INTERO-----DECIMALE"
15 PRINT:FORK=1TO8:READA$,A,B:GOSUB200:GOSUB300
20 PRINTTAB(1)A$;TAB(8)SPC(11-LEN(A1$));A1$;
23 PRINTSPC(8-LEN(B1$))B1$".";B2$
25 NEXTK
30 END
200 REM ROUTINE FORMATO INTERI
203 A1$=MID$(STR$(A),2)
205 IFLEN(A1$)>3THEN220
210 IFLEN(A1$)>7THEN230
215 RETURN
220 X1$=LEFT$(A1$,LEN(A1$)-3)+"."
223 A1$=X1$+MID$(A1$,LEN(A1$)-2)
```

```
225 GOTO210
230 X1$=LEFT$(A1$,LEN(A1$)-7)+"."
233 A1$=X1$+MID$(A1$,LEN(A1$)-6):RETURN
300 B1$=MID$(STR$(INT(B)),2)
305 B2$=MID$(STR$(B-VAL(B1$)),3)
307 B2$=LEFT$(B2$+S$,5)
310 RETURN
1000 DATAAAAA,38450,26.707,BBBBB,4900,2.404
1005 DATACCCC,400002,310.02,DDD,256000,443.2
1010 DATAEEEEE,2000,20,FFF,100,80.465
1015 DATAGG,10000000,0,HHHHH,7860,235.796
```

LETTURA DALLA MEMORIA (MEMORY DUMP): il problema di tabulare è più semplice se i dati hanno una lunghezza nota e costante. Nell'esempio che segue, dati un indirizzo iniziale e uno finale di memoria, viene presentato il contenuto della memoria 8 byte per linea, con il seguente formato:

-XXXXX-AAAAAAAA-YY-YY-YY-YY-YY-YY-YY-YY-
dove:

- XXXXX è l'indirizzo del primo degli 8 byte,
- ogni A rappresenta il carattere ASCII stampabile rappresentato nel byte o un punto,
- le 8 coppie YY rappresentano il contenuto degli 8 byte espresso in esadecimale.

Il programma ES16 realizza una tabulazione.

```
1 REM ES16
5 REM MEMORY DUMP
10 H$="0123456789ABCDEF"
20 INPUT"INDIRIZZO INIZ.: ";A1
25 IF0>A1ORA1>65534THEN20
30 INPUT"INDIRIZZO FIN.: ";A2
35 IFA1>A2ORA2>65535THEN30
40 INPUT"OK ";R$:IFR$="N"THEN20
50 IFR$<>"S"THEN40
60 L=1:K=A1
70 K$=RIGHT$("0000"+MID$(STR$(K),2),5)
80 PRINTSPC(1)K$SPC(1)"■";
95 FORJ=0TO7:A=PEEK(K+J)
100 A$=".":IFA>31ANDA<128ORA>159THENA$=CHR$(A)
120 PRINTA$;
```

```
125 POKE212,0
130 NEXTJ:PRINT"■";
145 FORJ=0TO7:A=PEEK(K+J)
147 X1=(AAND240)/16+1:X2=(AAND15)+1
150 PRINT"▮▮"MID$(H$,X1,1)MID$(H$,X2,1);
160 NEXTJ:PRINT
170 L=L+1+(L=24)*24:IFL<>1THEN190
180 GETR$:IFR$<>CHR$(13)THEN180
190 K=K+8:IFK<=A2THEN70
200 GOTO20
```

Osserva la semplice tecnica usata per contare le linee stampate, in modo da interrompere la stampa (fino alla pressione di CR), evitando lo scrolling indesiderato.

GRAFICO DELLA FUNZIONE SIN(X): il programma ES17 è un piccolo esempio di grafica con i caratteri. Il grafico compare linea per linea, sfruttando lo scrolling dello schermo.

```
1 REM ES17
5 REMGRAFICO FUNZIONE SENO
7 P$=".▮▮":A$="*▮▮"
10 PRINT"▯"SPC(250)SPC(250)SPC(250)SPC(250);
15 PRINTTAB(13)"FUNZIONE  SENO"SPC(13);
20 FORK=1TO40:PRINT"=";:NEXTK
25 PRINT"ANG. -1---------------0---------------1"
30 FORK=0TO360STEP15
35 PRINTRIGHT$("  "+STR$(K),3)"▮▮▮▮";
40 T=22+SIN(K/180*π)*16
45 IFT>6THENPRINTTAB(6)P$;
50 IFT<22THEN80
55 IFT=22THENPRINTTAB(T)A$TAB(38)P$;:GOTO65
60 PRINTTAB(22)" ▮▮"TAB(T)A$;
63 IFT<38THENPRINTTAB(38)P$;
65 PRINT:NEXTK
70 GOTO70
80 PRINTTAB(T)A$TAB(22)"I▮▮"TAB(38)P$;:GOTO65
```

2.5 GRAFICA IN MODO CARATTERI

Ben 128 dei 256 caratteri visualizzabili sullo schermo (74 nel set maiuscolo/minuscolo) non sono nè alfanumerici nè simbolici, ma CARATTERI GRAFICI. Il loro uso appropriato può permettere di creare disegni e grafici di discreta qualità. Ci occupiamo qui di questo argomento, dopo aver chiarito il significato di alcuni termini.

- PIXEL: è la più piccola unità di informazione grafica visualizzabile sullo schermo. Comunemente si usa dire PUNTO, ma, talvolta, un PIXEL può essere composto da diversi PUNTI DELLO SCHERMO, il cui numero indica la DIMENSIONE DEL PIXEL.
- RISOLUZIONE: è il numero massimo di PIXEL visualizzabili sullo schermo.
- NUMERO COLORI: è il numero di colori disponibili, accompagnato da quello del massimo numero di colori diversi utilizzabili in un certo raggruppamento di punti.
- FORMA DEL PIXEL: non è detto che il PIXEL sia per forza rettangolare, quadrato o puntiforme (alcuni calcolatori visualizzano PIXEL di forma esagonale).

2.5.1 Grafica a bassissima risoluzione

In modo caratteri lo schermo è suddiviso in 25 linee di 40 caratteri ciascuna. Se utilizziamo un carattere come PIXEL, possiamo ottenere grafici con le seguenti caratteristiche:

- RISOLUZIONE: 25*40=1000 PIXEL
- NUM. COLORI: 1 su 16 per ogni PIXEL
- FORMA DEL PIXEL: RETTANGOLARE o carattere ASCII-CBM
- DIMENSIONI DEL PIXEL: 8*8=64 PUNTI dello schermo (1 carattere).

Per fare comparire un pixel in una determinata posizione è sufficiente una PRINT o una coppia di POKE (meglio perchè evitano lo scrolling) basate sulle coordinate della posizione. A questo scopo puoi scegliere di riferirti allo schermo in uno dei sistemi di coordinate riportati in Figura 2.3.

Quelli più utilizzati sono a) e b), e il secondo dei due è forse il più naturale per chi è abituato a tracciare grafici, in quanto coincide con il primo quadrante in coordinate cartesiane.

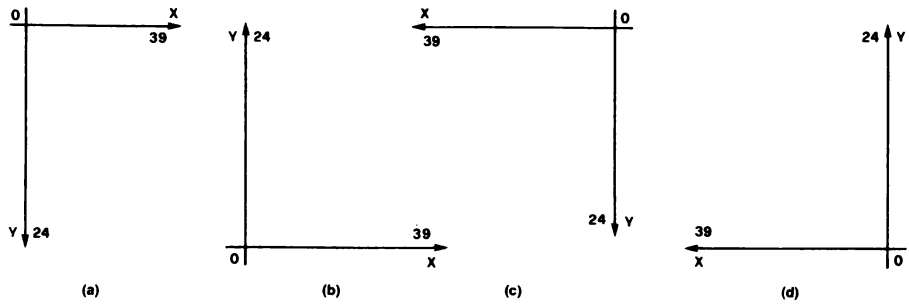


Figura 2.3 Sistemi di coordinate

Per le coordinate i limiti sono:
0 <=X <=39 e 0 <=Y <=24.
Una routine per disegnare un pixel sullo schermo può essere allora realizzata come nel programma esempio ES18.

```
1 REM ES18
10 REM PROVA ROUTINE
15 BV=1024:BC=55296:CH=160:CO=8
20 PRINT"□";:INPUT "COORDINATA X=";X
25 INPUT "COORDINATA Y=";Y:GOSUB1000
30 GOTO30
1000 REM ROUTINE DISEGNA PUNTO
1005 IFX<0ORY<0ORX>39ORY>24THEN1025
1010 A=40*(24-INT(Y))+INT(X)
1015 POKEBV+A,CH:POKEBC+A,CO
1020 RETURN
1025 PRINT"?FUORI DAL VIDEO","ERRORE":END
```

dove:
X,Y sono le coordinate del pixel nel sistema b),
BV è l'indirizzo base del video (1024 di norma),
BC è l'indirizzo della mappa colore (55296),
CH è il codice del carattere che si vuole usare come PIXEL (160 è il più comune),

CO è il colore del PIXEL (compreso tra 0 e 15).
Alla linea 1005 viene fatto un controllo sulla validità delle coordinate e in caso esse non siano nello schermo viene segnalato un errore e il programma si ferma. Puoi aggiungere controlli per i valori di CH e di CO per evitare il messaggio di errore: ?ILLEGAL QUANTITY ERROR, come nel programma ES19. Sulla cassetta trovi anche il programma ES19.1 che contiene una variazione rispetto ad ES19.

```
1 REM ES19
10 REM PROVA ROUTINE
15 BV=1024:BC=55296:CH=160:CO=8
20 PRINT"□";:INPUT "COORDINATA X=";X
25 INPUT "COORDINATA Y=";Y:GOSUB1000
30 GOTO30
1000 REM ROUTINE DISEGNA PUNTO CORRETTO
1005 XX=X:YY=Y
1010 IFXX>0ANDXX<40THEN1017
1015 XX=XX-(XX<0)*XX+(XX>39)*(XX-39):GOTO1010
1017 IFYY>0ANDYY<24THEN1025
1020 YY=YY-(Y=0)*YY+(YY>23)*(YY-23):GOTO1017
1025 A=40*(24-INT(YY))+INT(XX)
1030 POKEBV+A,CH:POKEBC+A,CO
1035 RETURN
```

Se sei assolutamente sicuro che i valori di X e Y non siano mai fuori dai limiti puoi eliminare la linea 1005.
Per cancellare un pixel puoi o ridisegnarlo nel colore dello sfondo o usare CH=32 e riscriverci sopra.
Per vedere un altro esempio di quanto è stato detto puoi provare il programma ES20.

```
1 REM ES20
5 REM GRAFICO DI UNA FUNZIONE
7 MX=39:MY=24:CC=646:CH=160
10 POKE53280,0:POKE53281,0:POKECC,1
12 DIMV(MX)
15 BV=1024:BC=55296:CO=2:KB=630:R$=CHR$(13)
40 F$="6"
45 DEF FNA(X)=6
```

```
50 PRINT"▯HAI GIA´ INSERITO LA FUNZIONE"
55 INPUT I$:IFI$="S"ORI$="SI"THEN100
60 INPUT"FUNZIONE:";F$:IFF$=""THEN60
65 IFLEN(F$)>65THENPRINT"TROPPO LUNGA":GOTO60
70 POKECC,0:PRINT"▯▯▯40 F$="CHR$(34)F$CHR$(34)
75 PRINT"45 DEF FNA(X)="F$R$"RUN▯";
80 E$=R$+R$+R$:GOSUB1500:END
100 REM
200 INPUT"▯ESTREMI (X1,X2):";X1,X2
205 IF X1>=X2 THEN200
210 PX=(X2-X1)/MX:E$="G⌐500"+R$
215 PRINT"▯STO C A L C O L A N D O !"
220 GOSUB1500:MI=FNA(X1):MA=MI
225 FORK=0TOMX:A=FNA(X1+K*PX):V(K)=A
230 IFA>MATHENMA=A:GOTO240
235 IFA<MITHENMI=A
240 NEXT
245 PY=(MA-MI)/MY
246 IFPY=0THENSTOP
250 PRINT"▯PREPARAZIONE G R A F I C O !"
255 FORK=0TOMX:V(K)=(V(K)-MI)/PY:NEXT
260 PRINT"▯";:FORK=0TOMX:X=K:Y=V(K):GOSUB1000
265 NEXT:GOSUB1520
268 GETA$:IFA$=""THEN268
270 PRINT"*** "F$" ***";:GOSUB1520
275 GETA$:IFA$=""THEN275
280 PRINT"▯▯ 1 - TORNA AL GRAFICO"
282 PRINT" 2 - CAMBIA GLI ESTREMI"
284 PRINT" 3 - CAMBIA LA FUNZIONE"
298 PRINT" 0 - TERMINA IL PROGRAMMA"
300 INPUT"▯   - COSA SCEGLI";I$
305 IFI$<"0"ORI$>"3"THEN280
310 ON VAL(I$)GOTO260,200,50
315 END
500 PRINT"▯ATTENZIONE LA FUNZIONE NON E´"
505 PRINT"DEFINITA CORRETTAMENTE"
510 PRINT"NELL´INTERVALLO:"X1"-"X2
580 PRINT"PREMI [F1] PER RIDEFINIRLA"
585 GOSUB1520
590 GETA$:IFA$<>"■"THEN590
595 GOTO50
999 STOP
```

```
1000 REM DISEGNA PUNTO
1005 IFX<0ORY<0ORINT(X)>39ORINT(Y)>24THEN1050
1010 A=40*(24-INT(Y))+INT(X)
1015 POKEBV+A,CH:POKEBC+A,CO
1020 RETURN
1050 PRINT"?FUORI DAL VIDEO","ERRORE":END
1500 REM ON ERROR GOTO
1501 REM METTE IN ON
1505 FORK=1TOLEN(E$)
1510 POKEKB+K,ASC(MID$(E$,K,1)):NEXT
1515 POKE198,LEN(E$):RETURN
1518 REM METTE IN OFF
1520 POKE198,0:RETURN
```

Nel TRACCIARE LINEE o altri grafici, se le coordinate di schermo di due punti differiscono per più di un'unità, i due punti appaiono separati. Se desideri vedere i punti uniti, devi tracciare linee da un punto di coordinate X1,Y1 a un punto di coordinate X2,Y2, con differenza di coordinate minore di uno.

Si possono presentare 4 casi:

a) I due punti si trovano sulla stessa linea orizzontale, cioè Y1=Y2; una routine può essere quella riportata nell'esempio ES22.

```
1 REM ES22
7 BV=1024:BC=55296:CH=160:CO=8
10 INPUT"X1= ";X1
20 INPUT"X2= ";X2
25 INPUT"Y1= ";Y1
27 Y2=Y1
30 PRINT"□":GOSUB1050
40 GOTO40
1050 REM LINEA ORIZZONTALE: Y1=Y2
1055 IFX1*Y1<0ORX2*Y2<0THENSTOP
1057 IFX1>39ORY1>24ORX2>39ORY2>24THENSTOP
1060 IFX1>X2THENT=X1:X1=X2:X2=T
1065 A=40*(24-Y1)+X1-1
1070 FORK=1TO(X2-X1+1):POKEBV+A+K,CH
1073 POKEBC+A+K,CO:NEXTK
1075 RETURN
```

b) I due punti si trovano sulla stessa linea verticale, cioè X1=X2; come nel programma esempio ES23.

```
1 REM ES23
7 BV=1024:BC=55296:CH=160:CO=8
10 INPUT"X1= ";X1
20 INPUT"Y1= ";Y1
25 INPUT"Y2= ";Y2
27 X2=X1
30 PRINT"□":GOSUB1050
40 GOTO40
1050 REM LINEA VERTICALE: X1=X2
1055 IFX1*Y1<0ORX2*Y2<0THENSTOP
1057 IFX1>39ORY1>24ORX2>39ORY2>24THENSTOP
1060 IFY1>Y2THENT=Y1:Y1=Y2:Y2=T
1065 A=40*(24-Y1)+X1-1
1070 FORK=0TO(Y2-Y1):POKEBV+A-K*40,CH
1073 POKEBC+A-K*40,CO:NEXTK
1075 RETURN
```

c) I due punti si trovano su una retta a 45 gradi, cioè ABS(X2-X1)=ABS(Y2-Y1), come nel programma esempio ES24.

```
1 REM ES24
7 BV=1024:BC=55296:CH=160:CO=8
10 INPUT"X1= ";X1
20 INPUT"X2= ";X2
25 INPUT"Y1= ";Y1
27 INPUT"Y2= ";Y2
28 IFABS(X2-X1)<>ABS(Y2-Y1)THENSTOP
30 PRINT"□":GOSUB1050
40 GOTO40
1000 REM ROUTINE DISEGNA PUNTO
1005 IFX<0ORY<0ORX>39ORY>24THEN1025
1010 A=40*(24-INT(Y))+INT(X)
1015 POKEBV+A,CH:POKEBC+A,CO
1020 RETURN
1025 PRINT"?FUORI DAL VIDEO","ERRORE":END
1050 REM LINEA A 45 GRADI:ABS(X2-X1)=ABS(Y2-Y1)
```

```
1055 IFX1*Y1<0ORX2*Y2<0THENSTOP
1057 IFX1>39ORY1>24ORX2>39ORY2>24THENSTOP
1070 X=X1:FORK=Y1TOY2STEPSGN(Y2-Y1):Y=K
1073 GOSUB1005:X=X+1:NEXTK
1075 RETURN
```

d) I due punti non sono allineati nei modi precedenti.

Sulla cassetta trovi registrati anche i 3 programmi ES22.1, ES23.1 e ES24.1, che sono variazioni dei programmi precedenti.
Il caso d) è il più comune e il più difficile da risolvere. Infatti non basta tracciare una linea, ma si cerca di tracciare una buona linea, cioè una linea con le seguenti caratteristiche:

- inizio e fine nei punti dati,
- il più possibile diritta,
- a densità costante,
- senza discontinuità.

In sintesi il problema consiste nel trovare il numero di punti necessario per tracciare la linea, tenendo conto della risoluzione e dell'approssimazione ottenibile. Il calcolo più semplice consiste nello scegliere come numero dei punti il maggiore tra i valori ABS(X2-X1) e ABS(Y2-Y1), dividere entrambi gli intervalli per questo numero per ricavare l'incremento per ogni coordinata (sarà 1 o minore di 1) e tracciare la linea punto per punto.

```
1 REM ES25
5 BV=1024:BC=55296:CH=160:C0=8
10 PRINT"COORDINATE DUE PUNTI"
15 INPUT"COORDINATA X1=";X1
16 IFX1<0ORX1>39THEN15
20 INPUT"COORDINATA Y1=";Y1
21 IFY1<0ORY1>23THEN20
25 INPUT"COORDINATA X2=";X2
26 IFX2<0ORX2>39THEN25
30 INPUT"COORDINATA Y2=";Y2
31 IFY2<0ORY2>23THEN30
40 GOSUB1200
50 GOTO50
1000 REM PLOT
1010 A=40*(24-INT(Y))+INT(X)
```

```
1015 POKEBV+A,CH:POKEBC+A,CO
1020 RETURN
1200 REM DRAW DA X1,Y1 A X2,Y2
1205 L=ABS(X2-X1)
1206 IF ABS(Y2-Y1)>L THEN L=ABS(Y2-Y1)
1210 DX=(X2-X1)/L:DY=(Y2-Y1)/L
1215 X=X1+.5:Y=Y1+.5
1220 FORK=1TOL:GOSUB1000:X=X+DX:Y=Y+DY:NEXTK
1225 RETURN
```

Il grafico tracciato con il programma ES25 presenta molte irregolarità, che dipendono dalle dimensioni del pixel.
Il programma ES25.1 disegna una linea sovrapponendola a una griglia.

```
1 REM ES25.1
5 BV=1024:BC=55296:CH=160:C0=8
20 X1=4:Y1=13:X2=10:Y2=21:PRINT"▯";
25 PRINT"▮▮▮▮▮";
30 FORK=1TO10:PRINT"▮++++++++++"SPC(30);:NEXT
40 GOSUB1200
50 GOTO50
1000 REM PLOT
1010 A=40*(24-INT(Y))+INT(X)
1015 POKEBV+A,CH:POKEBC+A,CO
1020 RETURN
1200 REM DRAW DA X1,Y1 A X2,Y2
1205 L=ABS(X2-X1)
1206 IF ABS(Y2-Y1)>L THEN L=ABS(Y2-Y1)
1210 DX=(X2-X1)/L:DY=(Y2-Y1)/L
1215 X=X1+.5:Y=Y1+.5
1220 FORK=1TOL:GOSUB1000:X=X+DX:Y=Y+DY:NEXTK
1225 RETURN
```

Seguono alcuni programmi esempio: ES26, ES27, ES28.

```
1 REM ES26
10 REM SPIRALE
20 BV=1024:BC=55296:CH=81:CO=8
30 X1=20:Y1=12:C=0:PRINT"▯";
40 FORJ=1TO25:X2=X1+J*COS(C*π/2)
```

```
50 Y2=Y1+J*SIN(C*π/2):GOSUB1200
60 C=C+1+4*(C=3):X1=X2:Y1=Y2
70 NEXT
80 GOTO80
1000 REM PUNTO
1005 IFX<0ORY<0ORINT(X)>39ORINT(Y)>24THEN1050
1010 A=40*(24-INT(Y))+INT(X)
1015 POKEBV+A,CH:POKEBC+A,CO
1020 RETURN
1050 PRINT"?OUT OF SCREEN","ERROR":END
1200 REM DISEGNA
1205 L=ABS(X2-X1)
1207 IF ABS(Y2-Y1)>L THEN L=ABS(Y2-Y1)
1210 DX=(X2-X1)/L:DY=(Y2-Y1)/L
1215 X=X1+.5:Y=Y1+.5
1220 FORK=1TOL:GOSUB1000:X=X+DX
1223 Y=Y+DY:NEXTK:GOSUB1000
1225 RETURN
```

```
1 REM ES27
5 REM POLIGONO
10 BV=1024:BC=55296:CH=160
20 INPUT"NUM LATI (>0) ";NL:IFNL<1THEN20
30 INPUT"LUNGHEZZA(>0) ";LE:IFLE<0THEN30
40 Y1=0:X1=20-LE/2:GOSUB650
50 INPUT"ANCORA   (S/N) ";R$
55 IFR$="S"THEN20
60 IFR$<>"N"THEN50
70 END
650 REM POLIGONI
655 AN=0:FORJ=1TONL
660 X2=X1+LE*COS(AN):Y2=Y1+LE*SIN(AN)
665 GOSUB1200
670 X1=X2:Y1=Y2:AN=AN+2*π/NL
675 NEXT:RETURN
1000 REM PUNTI
1005 IFX<0ORY<0ORINT(X)>39ORINT(Y)>24THEN1050
1010 A=40*(24-INT(Y))+INT(X)
```

```
1015 POKEBV+A,CH:POKEBC+A,CO
1020 RETURN
1050 PRINT"?OUT OF SCREEN","ERROR":END
1200 REM LINEE
1205 L=ABS(X2-X1)
1206 IF ABS(Y2-Y1)>L THEN L=ABS(Y2-Y1)
1210 DX=(X2-X1)/L:DY=(Y2-Y1)/L
1215 X=X1+.5:Y=Y1+.5
1220 FORK=1TOL:GOSUB1000:X=X+DX
1223 Y=Y+DY:NEXTK:GOSUB1000
1225 RETURN
```

```
1 REM ES28
5 REM CIRCOLI
10 BV=1024:BC=55296:CH=160:CO=6:XC=20:YC=12
15 INPUT"RAGGIO (1-12) ";R:IFR<1ORR>12THEN15
20 INPUT"COLORE (0-15) ";CO
23 IFCO<0ORCO>15THEN20
25 PRINT"■";:GOSUB600
30 INPUT"■ANCORA (S/N)";R$:IFR$="S"THEN15
35 IFR$<>"N"THEN30
40 STOP
600 REM CIRCOLO
605 FORT=0.075TO2*πSTEP.075
610 X=XC+R*SIN(T):Y=YC+R*COS(T)
615 GOSUB1000:NEXT:RETURN
1000 REM PUNTI
1005 IFX<0ORY<0ORINT(X)>39ORINT(Y)>24THEN1050
1010 A=40*(24-INT(Y))+INT(X)
1015 POKEBV+A,CH:POKEBC+A,CO
1020 RETURN
1050 PRINT"?OUT OF SCREEN","ERROR":END
```

Sulla cassetta è registrato anche il programma ES27.1, una variazione di ES27.

2.5.2 Grafica a bassa risoluzione

Dei 74 caratteri grafici comuni ai due set di caratteri del COMMODORE 64 ce ne sono 16 che possono essere utilizzati per moltiplicare per 4 la risoluzione dello

schermo. In questo modo ogni pixel sarà un quarto di un carattere. Si ha:

- RISOLUZIONE: 50*80=4000 pixel,
- NUM. COLORI: 1 su 16 per ogni gruppo di 4 pixel,
- FORMA DEL PIXEL: rettangolare,
- DIMENSIONI DEL PIXEL: 4*4 punti (1/4 di carattere).

I 16 caratteri utilizzati sono riportati nella Figura 2.4; essi mostrano le 16 possibili combinazioni di punti per ogni carattere.

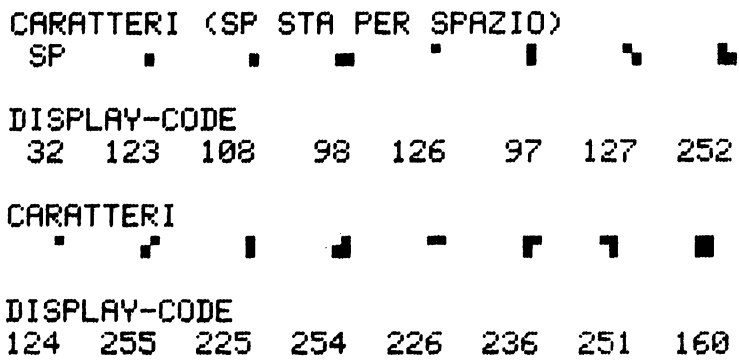


Figura 2.4 Caratteri per GRAFICA a BASSA RISOLUZIONE

La procedura per disegnare un pixel differisce da quella per la grafica a bassissima risoluzione in due aspetti in particolare:

A) per poter disegnare un pixel bisogna sapere se ci sono altri pixel nella zona carattere (che può contenere 4 pixel), dove esso dovrà comparire;

B) il carattere da scrivere dipende sia dalla posizione del pixel da disegnare, sia da quello che esiste già sullo schermo.

Per risolvere il problema A) è sufficiente, una volta calcolato l'indirizzo del carattere sullo schermo che conterrà il PIXEL da disegnare, confrontare il D-CODE del carattere, già presente in quella locazione, con i 16 riportati nella Figura 2.4. Se il confronto è positivo ci sono già pixel disegnati nella zona carattere, altrimenti il carattere eventualmente presente è alfanumerico o grafico e allora si può scegliere se sovrascrivere oppure non disegnare il pixel. La routine che controlla se c'è già qualche pixel presente sullo schermo, può essere come indicato in ES31, da 1250 a 1265.

```
1 REM ES31
3 REM CANCELLAZIONE PUNTI
5 DIMD(15),X(100),Y(100)
10 RESTORE:FORK=0TO15:READD(K):NEXT
11 DATA32,123,108,98,126,97,127,252,124
12 DATA255,225,254,226,236,251,160
15 BV=1024:BC=55296:CO=0:NP=0:PRINT"□";
20 Q=INT(RND(1)*100)+3
30 FORJ=1TO100:X(J)=INT(RND(Q)*80)
35 Y(J)=INT(RND(Q)*50):NEXT
40 FORJ=1TO100:X=X(J):Y=Y(J):GOSUB1300:NEXT
45 FORJ=1TO100:X=X(J):Y=Y(J):GOSUB1350:NEXT
50 GOTO20
1250 REM SITUAZIONE POSIZIONE CARATTERE
1255 P1=PEEK(BV+A):H=16
1260 FORK=0TO15:IFP1=D(K)THENH=K:K=15
1265 NEXT:RETURN
1300 REM PUNTI
1305 IFX<0ORY<0ORINT(X)>79ORINT(Y)>49THEN1340
1310 A=40*(24-INT(Y/2))+INT(X/2)
1315 GOSUB1250:IFH=16THENH=0:IFNPTHENRETURN
1320 CH=D(H OR (2↑(X AND 1)*4↑(Y AND 1)))
1325 POKEBV+A,CH:POKEBC+A,CO
1330 RETURN
1340 PRINT"?OUT OF SCREEN  ERROR":END:RETURN
1350 REM CANCELLA PUNTI
1355 IFX<0ORY<0ORINT(X)>79ORINT(Y)>49THEN1340
1360 A=40*(24-INT(Y/2))+INT(X/2)
1365 GOSUB1250:IFH=16THENH=0:IFNPTHENRETURN
1370 CH=D(HAND(15-(2↑(X AND 1)*4↑(Y AND 1))))
1375 POKEBV+A,CH:POKEBC+A,CO
1380 RETURN
```

Il vettore D è stato inizializzato nelle prime linee del programma, da 5 a 12, usando READ e DATA.
Nel programma ES31 A è l'indirizzo della posizione carattere del video dove dovrà comparire il pixel, H contiene 16 se il carattere della A-esima posizione dello schermo non è compreso tra quelli indicati per questo tipo di grafica, un numero tra 0 e 15 se il carattere è, invece, tra quelli.
I valori limite per le coordinate sono:

$0 <= X <= 79$ e $0 <= Y <= 49$
e, utilizzando i sistemi di coordinate prima descritti, l'indirizzo della mappa video del carattere da modificare si calcola come:
A=40*INT(Y/2)+INT(X/2) nel sistema a), oppure:
A=40*(24-INT(Y/2))+INT(X/2) nel sistema b).
Il pixel dovrà comparire in una delle 4 posizioni possibili, e, a seconda del carattere letto dallo schermo, si dovrà ottenere il codice del carattere da riscrivere (sovrapporre).
Per risolvere il problema B), dopo aver chiamato la routine contenuta in ES31 nelle linee 1250-1265, si può calcolare CH come alla linea 1320 di ES31.
Questo perchè i dati nella matrice D sono ordinati in modo che il carattere nella posizione $0 <= N <= 15$ risulti la composizione dei 4 caratteri base ai quali sono associati i valori indicati in Figura 2.5.

VALORE IN D(N)	CARATTERE	N
D(1)=123	▘	1
D(2)=108	▝	2
D(4)=126	▖	4
D(8)=124	▗	8

Figura 2.5 Caratteri base per GRAFICA a BASSA RISOLUZIONE con coordinate di tipo (b)

Ad esempio i caratteri, il cui codice è in D(13) o in D(9), sono indicati in Figura 2.6.

VALORE IN D(N)	CARATTERE	N	COMPONENTI
D(13)=236	▛	13=8+4+1	▘▝▗
D(9)=255	▚	9=8+1	▘▗

Figura 2.6 Esempi di caratteri del vettore D

La routine inserita in ES31, da 1300 a 1340, serve per disegnare un pixel nella posizione X,Y; se NP < > 0 non sovrascrive, se sullo schermo non esiste un carattere grafico, se NP=0 disegna comunque il pixel. Questo sottoprogramma è una variazione dell'analogo per la bassissima risoluzione.

Per preparare la routine che permette di CANCELLARE un pixel, come puoi vedere da ES31, è sufficiente copiare le linee dalla 1300 alla 1340 nelle linee dalla 1350 alla 1380 modificando la linea 1320, che diventa la 1370.

Sulla cassetta sono registrati due programmi che tracciano linee in bassa risoluzione, ES32 e ES33, che non sono qui listati.

ATTENZIONE: se utilizzi il sistema di coordinate che pone il punto 0,0 nell'angolo in alto a sinistra, devi riordinare i valori nelle linee DATA, in modo che ai 4 caratteri base siano associati i valori indicati in Figura 2.7.

VALORE IN D(N)	CARATTERE	N
D(1)=126	▘	1
D(2)=124	▝	2
D(4)=123	▖	4
D(8)=108	▗	8

Figura 2.7 Caratteri base per GRAFICA a BASSA RISOLUZIONE con coordinate di tipo (a)

Per tracciare linee in bassa risoluzione non fa molta differenza se i punti sono o meno allineati, proprio per il problema di dover controllare se ci sono già dei punti sullo schermo. L'unica modifica da apportare alla routine che disegna dell'esempio ES25 è in questo caso:

1220 FOR K=1 TO L:GOSUB 1300:X=X+DX:Y=Y+DY:NEXT K

Ora puoi adattare gli esempi riportati per la bassissima risoluzione alla bassa risoluzione. L'unico problema è che se disegni un pixel di un certo colore, cambi il colore degli altri tre pixel che si trovano nella stessa posizione carattere. E' difficile trovare una soluzione soddisfacente.

Puoi provare anche il programma ES34 che segue, che fa rimbalzare un pixel su tutto il video.

```
1 REM ES34
5 DIMD(15),X(100),Y(100)
10 RESTORE:FORK=0TO15:READD(K):NEXT
11 DATA32,123,108,98,126,97,127,252,124,255
12 DATA225,254,226,236,251,160
15 BV=1024:BC=55296:CO=6:NP=0:PRINT"□";
20 MX=79:MY=49:POKE53281,1
100 XN=INT(RND(0)*20)+20
105 YN=INT(RND(0)*10)+15
110 DX=1:DY=-1
```

```
120 FORF=0TO1:F=0
125 XO=XN:YO=YN
130 XN=XN+DX:IFXN=0ORXN=MXTHENDX=-DX
135 YN=YN+DY:IFYN=0ORYN=MYTHENDY=-DY
140 X=XN:Y=YN:GOSUB1300:FORT=0TO10:NEXT
145 X=XN:Y=YN:GOSUB1350
150 NEXT
1250 REM PUNTO
1255 P1=PEEK(BV+A):H=16
1260 FORK=0TO15:IFP1=D(K)THENH=K:K=15
1265 NEXT:RETURN
1300 REM DISEGNA
1305 IFX<0ORY<0ORINT(X)>79ORINT(Y)>49THEN1390
1310 A=40*(24-INT(Y/2))+INT(X/2)
1315 GOSUB1250:IFH=16THENH=0:IFNPTHENRETURN
1320 CH=D(H OR (2↑(X AND 1)*4↑(Y AND 1)))
1325 POKEBV+A,CH:POKEBC+A,CO
1330 RETURN
1350 REM CANCELLA
1355 IFX<0ORY<0ORINT(X)>79ORINT(Y)>49THEN1390
1360 A=40*(24-INT(Y/2))+INT(X/2)
1365 GOSUB1250:IFH=16THENH=0:IFNPTHENRETURN
1370 CH=D(HAND(15-(2↑(X AND 1)*4↑(Y AND 1))))
1375 POKEBV+A,CH:POKEBC+A,CO
1380 RETURN
1390 PRINT"?FUORI DAL VIDEO  ERRORE":END
```

2.5.3 Istogrammi

Per ottenere buoni istogrammi possiamo usare caratteri grafici appartenenti ai 4 gruppi riportati nella Figura 2.8.

CAR.	TASTI	D-CODE	CAR.	TASTI	D-CODE
	CBM+T	99	_	CBM+@	100
‾	CBM+Y	119	▬	CBM+P	111
▬	CBM+U	120	▬	CBM+O	121
▬	RVS/CBM+I	226	▬	CBM+I	98
▬	RVS/CBM+O	249	▬	RVS/CBM+U	248
▬	RVS/CBM+P	239	▬	RVS/CBM+Y	247
▬	RVS/CBM+@	228	▬	RVS/CBM+T	227
▌	CBM+G	101	▌	CBM+M	103
▌	CBM+H	116	▌	CBM+N	106
▌	CBM+J	117	▌	CBM+L	118
▌	CBM+K	97	▌	RVS/CBM+K	225
▌	RVS/CBM+L	246	▌	RVS/CBM+J	245
▌	RVS/CBM+N	234	▌	RVS/CBM+H	244
▌	RVS/CBM+M	231	▌	RVS/CBM+G	229

Figura 2.8 Caratteri grafici usati per tracciare ISTOGRAMMI

In questi gruppi il carattere è considerato come composto da 8 barrette, elementi orizzontali o verticali di 8 punti ciascuno. Utilizzando questi caratteri grafici insieme allo spazio e allo spazio inverso (rettangolo vuoto o rettangolo pieno), possiamo creare sullo schermo dei grafici a barre (istogrammi). La risoluzione orizzontale è di 40*8=320 barrette e quella verticale di 25*8=200 barrette.
Per comprendere il metodo usato per creare una barra immagina di aver calcolato che essa debba essere lunga 263 elementi. Per disegnarla è sufficiente scrivere INT(263/8) caratteri spazio inverso, e, supponendo che la barra debba essere disegnata da sinistra a destra, il carattere CBM-M in campo inverso, che contiene 263-INT(263/8)*8=263-32*8=263-256=7 barrette.
Per fare comparire la barra sul video si possono usare almeno due procedimenti diversi:

- per intero, cioè stampare gli spazi in campo inverso e, eventualmente, il necessario carattere di completamento;

● gradualmente, cioè stampare uno sull'altro di seguito i caratteri appartenenti allo stesso gruppo per dare l'impressione di una barra che cresce.
In entrambi i casi possono essere usati sia PRINT che POKE; come vediamo nel programma ES35.

```
1 REM ES35
3 REM ISTOGRAMMI
5 DIM A$(7):RESTORE
10 FORK=0TO7:READA,B:A$(K)=CHR$(B)
13 IFATHENA$(K)="▨"+A$(K)+"▬"
15 NEXTK
20 PRINT"▯▯";:FORK=1TO10:READA
25 PRINT"BARRA DA"A":":N=A:GOSUB3000:PRINT
30 NEXTK:END
35 END
40 DATA0,32,0,180,0,165,0,181,0,161,1
41 DATA182,1,170,1,167
42 REM DATI PER BARRE
44 DATA 1,2,7,18,57,101,175,206,320,295
3000 REM BARRE SX TO DX
3005 N1=INT(N/8)
3010 IFN1THENPRINT"▨";
3013 FORI=1TON1:PRINTCHR$(32);:NEXT:PRINT"▬";
3015 N1=N-N1*8
3020 PRINTA$(N1);:RETURN
```

Per vedere crescere più lentamente le barre puoi provare il programma ES35.1 che segue.

```
1 REM ES35.1
3 REM ISTOGRAMMI
5 DIM A$(7):RESTORE
10 FORK=0TO7:READA,B:A$(K)=CHR$(B)
13 IFATHENA$(K)="▨"+A$(K)+"▬"
15 NEXTK
20 PRINT"▯▯";:FORK=1TO10:READA
25 PRINT"BARRA DA"A":":N=A:GOSUB3000:PRINT
30 NEXTK:END
35 REM DATI PER A$()
40 DATA0,32,0,180,0,165,0,181,0,161,1
```

```
41 DATA182,1,170,1,167
42 REM DATI PER BARRE
44 DATA 1,2,7,18,57,101,175,206,320,295
3000 REM BARRE SX TO DX
3005 N1=INT(N/8)
3010 IF N1=0THEN3025
3015 FORI=1TON1:FORJ=0TO7:PRINTA$(J)"▮▮";
3017 FORT=1TO20:NEXT
3020 NEXT:PRINT"▨ ▬";:NEXT
3025 N1=N-N1*8
3030 FORJ=0TON1:PRINTA$(J)"▮▮";
3033 FORT=1TO20:NEXT
3035 NEXT:PRINT"▮▮";:RETURN
```

Per creare istogrammi verticali le routine sono quasi identiche: in quelle con la PRINT bisogna posizionare il carattere successivo di sopra e non di fianco all'ultimo carattere disegnato. Inoltre se fai iniziare l'istogramma in basso a destra provocherai lo scrolling. Usando le POKE, invece, cambia solo il modo di variare PV. La routine viene modificata come indicato in ES36.

```
1 REM ES36
5 REM ISTOGRAMMI VERTICALI
10 FORK=0TO7:READA,B:A$(K)=CHR$(B)
15 IFATHENA$(K)="▨"+A$(K)+"▬"
20 NEXT
25 PRINT"▯";:FORK=1TO8:READA
30 PRINT"▨"SPC(255)SPC(255)SPC(255)SPC(195);
35 PRINTTAB(1+4*K+(A>9)+(A>99))A"▮▮▮▮";
40 PRINTTAB(1+4*K)"▯";:N=A:GOSUB3000:NEXT
43 GOTO43
44 REM DATI PER A$()
45 DATA0,32,0,164,0,175,0,185,0,162,1
46 DATA184,1,183,1,163
47 REM
48 REM DATI PER IL GRAFICO
49 DATA100,27,180,39,49,87,131,90
3000 REM ROUTINE BARRE BOT TO TOP VELOCE
3005 N1=INT(N/8)
3010 IFN1=0THEN3020
```

```
3015 PRINT"■";:FORI=1TON1:PRINT" ■";:NEXT
3020 PRINT"■";:N1=N-N1*8
3025 PRINTA$(N1);:RETURN
```

Sulla cassetta sono registrati anche i programmi ES36.1 e ES36.2; essi sono una variazione di ES36. Il primo fa crescere le barre più lentamente, il secondo le traccia colorate.

Non includiamo anche tutte le altre routine possibili, in quanto occuperebbero molte pagine, e, inoltre, esse sono molto semplici da ricavare basandosi sugli esempi esposti.
Come al solito abbiamo una limitazione nel numero dei colori; ogni gruppo di 8 barrette può contenere elementi di un colore a scelta tra i 16 possibili o del colore dello sfondo. A volte può essere utile disegnare grafici con barre al di sopra e al di sotto di una linea orizzontale di demarcazione, o a sinistra e a destra di una linea verticale. Per ottenerli bisogna avere l'accortezza di fare coincidere le origini delle coppie di barre e usare due colori diversi.

Segue un esempio di questo metodo nel programma ES37.

```
1 REM ES37
5 REM ISTOGRAMMI A 2 COLORI
10 PRINT"■";
20 DIMSU$(7),GIU$(7)
30 FORK=0TO7:READSU$(K),GIU$(K):NEXT
40 FORK=1TO30:READ V:CO=-2*(V>0)
50 PRINT"■"SPC(255)SPC(185)SPC(4+K-40*(V<0));
55 GOSUB100
60 NEXT
70 PRINT"■";:POKE646,6
80 GOTO80
100 REM ISTO. SU E GIU
105 CH=INT(ABS(V)/8):P=ABS(V)-CH*8
106 POKE646,CO
110 IFV<0THEN130
115 REM ISTOGRAMMA IN SU
120 IF CH THEN FORJ=1TOCH:PRINT"■ ■";:NEXT
125 PRINTSU$(P);:RETURN
130 REM ISTOGRAMMA IN GIU
```

```
135 IF CH THEN FORJ=1TOCH:PRINT"▝ ▟▙";:NEXT
140 PRINTGIU$(P);:RETURN
8000 DATA" ","▗","▄","▜","▟","▜","▟"
8003 DATA"▜","▟","▝▟"
8005 DATA"▝▜","▝▟","▗▜","▗▟","▝▜","▝▟"
8010 DATA1,2,3,10,20,-6,-4,-1,-50,+80
8015 DATA0,11,-10,-30,-55,-23,11,90,-96,3
8020 DATA49,47,-50,-25,77,80,-80,-5,-1,1
```

2.6 ESEMPI DI USO DEI CARATTERI GRAFICI

Seguono alcuni programmi o sottoprogrammi esempio che sfruttano i caratteri grafici disponibili.

QUASI ANIMAZIONE IN BASIC: usando un sistema simile a quello usato per gli istogrammi, disegnamo un quadratino che si muove sul video lungo una linea. Usiamo i caratteri riportati in Figura 2.9.

```
      COPPIE  DI  CARATTERI
▝  , ▝▌, ▝▌, ▝▌, ▝▝, ▌▝, ▌▝,  ▝

UN CARATTERE DIMINUISCE E L'ALTRO CRESCE
```

Figura 2.9 Coppie di caratteri da usare per produrre animazione

Il programma ES38 che segue fa muovere un quadratino sulla quarta linea dello schermo; come vedi il programma è abbastanza semplice.

```
5 REM ES38
10 DIMA(7)
12 S$="
15 FORK=0TO7:READA(K):NEXT
20 DATA101,116,117,97,246,234,231,160
30 PRINT"▚▚▚▚";S$;"▚";
```

```
35 BA=1144:H=10:C=0
40 FORK=0TO1:K=0
45 A=128+256*(A(C)>127)
50 POKE BA+H,A(C)+A
55 POKE BA+H+1+(1+H)*(H=39),A(C)
60 C=C+1+8*(C=7)
63 IFC=0THENPOKEBA+H,32:H=H+1+40*(H=39)
65 NEXT
```

Il calcolo di A permette di evitare l'uso di due vettori o di un vettore più lungo, dato che i due caratteri da stampare di seguito sono uno l'inverso dell'altro. La PRINT alla linea 20 pone nella mappa colore il codice colore del cursore in tutta la terza linea dello schermo. Un ciclo avrebbe occupato meno memoria. Il ciclo da 40 a 65 non ha mai termine.

TERMOMETRO: usiamo questa volta i 16 caratteri grafici che mostrano ognuno una barretta singola in ciascuna delle 8 possibili posizioni; essi sono riportati nella Figura 2.10.

CAR.	TASTI	D-CODE	CAR.	TASTI	D-CODE
▏	CBM+G	101	▔	CBM+T	99
▏	SHIFT+T	84	▔	SHIFT+E	69
▏	SHIFT+G	71	▔	SHIFT+D	68
▏	SHIFT+B	66	▔	SHIFT+C	67
▏	SHIFT+-	93	▔	SHIFT+*	64
▏	SHIFT+H	72	▁	SHIFT+F	70
▏	SHIFT+Y	89	▁	SHIFT+R	82
▏	CBM+M	103	▁	CBM+@	100

Figura 2.10 Caratteri a barretta singola

Nel programma ES39 usiamo le barrette verticali per simulare il movimento dell'aghetto di un termometro che riporta i dati termici relativi ad un ipotetico paziente (i dati sono prelevati da un DATA, ma l'ingresso potrebbe essere ottenuto da tastiera o anche collegando il calcolatore a un termometro reale).

```
1 REM ES39
5 REM TERMOMETRO
6 DIMA$(7):RESTORE:C$="▒"
7 FORK=0TO7:READA:A$(K)=CHR$(A):NEXT
8 DATA165,212,199,194,221,200,217,167
9 CN$="3 6  3 7  3 8  3 9  4 0  4 1  4 2  4 3"
10 DATA37,39,35,38.5,40.5,44,37.75,-1
13 S$="":FORK=1TO38:S$=S$+" ":NEXTK
15 PRINT"▒  "+CN$;
20 FORK=1TO8:PRINT"──┼─";:NEXT
25 PRINT"▒"S$"▒";
30 FORK=1TO8:PRINT"──┼─";:NEXT
35 BS=35.5:FS=43.05:RI=40:XP=0
40 RESTORE:FORK=1TO8:READA:NEXT
45 FORK=0TO1:K=0
50 READDA:IFDA=-1THENK=1:NEXT:GOTO40
55 X=(DA-BS)*RI:GOSUB100:XP=X:NEXT
60 END
100 REM SPOSTA AGO
105 IFX=XPTHENRETURN
110 IFXP<0THENXP=0
115 IFXP>303THENXP=303
120 FORJ=XPTOXSTEPSGN(X-XP):PRINT"▒▒▒▒";
125 IFJ>=0ANDJ<304THEN145
130 IFJ<0THENPRINT"▒"C$"▒"S$C$;:GOTO140
135 IFJ>303THENPRINTC$S$"▒"C$"▒";
140 J=X:FORL=1TO150:NEXT:NEXT:RETURN
145 PO=INT(J/8):CH=J-PO*8
150 PRINTC$LEFT$(S$,PO)A$(CH)LEFT$(S$,37-PO)C$;
155 NEXT:FORL=1TO150:NEXTL:RETURN
```

Il valore minimo rappresentabile è 35.5, il massimo 43.5. Il numero di posizioni possibili per l'ago è 38*8 (poichè il primo e l'ultimo carattere della linea sono occupati), cioè 304.
La differenza di temperatura tra una posizione dell'ago e la successiva è 0.025.

SCACCHIERE: ne puoi ottenere di tutte le dimensioni. Ti mostreremo tre esempi, ma ne potrai creare tanti altri. In Figura 2.11 sono riportati 3 tipi di scacchiera e seguono i 3 programmi per ottenerle: ES41.1, ES41.2 e ES41.3.

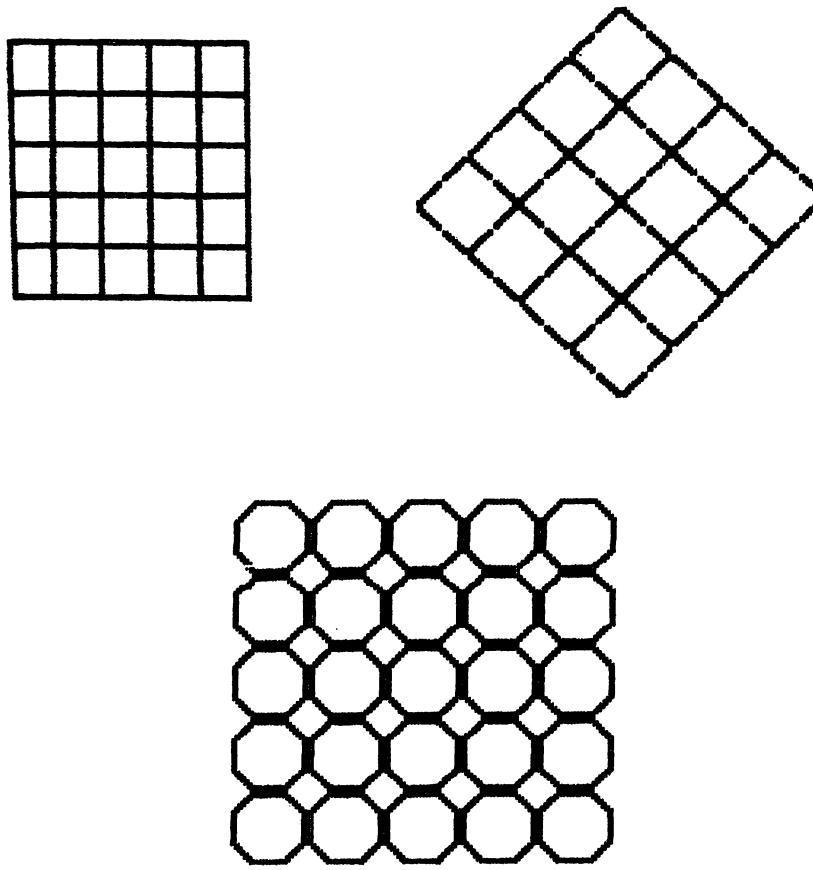


Figura 2.11 Scacchiere ottenute con i programmi ES41.1, ES41.2 E ES41.3

```
1 REM ES41.1
5 REM SCACCHIERA TIPO 1
10 INPUT"□NUMERO QUADRATI/LINEA (MAX 12)";NL
13 NL=INT(NL)
15 NL=NL-NL*(NL<0)+(12-NL)*(NL>12):PRINT"□";
20 PRINT" r";
```

```
23 IFNL>1THENFORK=1TONL-1:PRINT"─┬";:NEXT
25 PRINT"─┐"
30 IFNL=1THEN60
40 FORJ=1TONL-1
45 FORK=1TONL:PRINT"│ ";:NEXT:PRINT"│"
50 PRINT"├";:FORK=1TONL-1:PRINT"─┼";:NEXT
53 PRINT"─┤"
55 NEXT
60 FORK=1TONL:PRINT"│ ";:NEXT:PRINT"│"
65 PRINT"└";
67 IFNL>1THENFORK=1TONL-1:PRINT"─┴";:NEXT
70 PRINT"─┘";
75 GOTO75
```

```
1 REM ES41.2
5 REM SCACCHIERA TIPO 2
10 PRINT"[illegible]";
20 FORK=1TO4
25 FORJ=1TOK
30 PRINTTAB(20-2*K+4*(J-1))"/\";
35 NEXT:PRINT
40 FORJ=1TOK
45 PRINTTAB(19-2*K+2*(J-1))"/  \";
50 NEXT:PRINT:NEXT
60 FORK=4TO1STEP-1
65 FORJ=1TOK
70 PRINTTAB(19-2*K+2*(J-1))"\  /";
75 NEXT:PRINT
80 FORJ=1TOK
85 PRINTTAB(20-2*K+4*(J-1))"\/";
90 NEXT:PRINT:NEXT
95 GOTO95
```

```
1 REM ES41.3
5 REM SCACCHIERA TIPO 3
10 INPUT"[illegible]RIGHE (MAX 8)";R%
12 IFR%<1ORR%>8THEN10
15 INPUT"[illegible]COLONNE (MAX 13)";C%
```

```
17 IFC%<1ORC%>13THEN15
20 PRINT"";
25 FORJ=1TOR%
30 FORK=1TOC%:PRINT"/\";:NEXT:PRINT
40 FORK=1TOC%:PRINT"|  |";:NEXT:PRINT
50 FORK=1TOC%:PRINT"\_/";:NEXT:PRINT
55 NEXT
60 GOTO60
```

TASTIERA: Il programma ES42 disegna una tastiera con 19 tasti bianchi e 14 tasti neri, coprendo due ottave. Può essere uno spunto se sei appassionato di musica, per creare anche programmi didattici, ad esempio per visualizzare gli accordi.

```
1 REM ES42
705 REM TASTIERA
710 A$="| ▗ ▄ ▗ ▄ | ▗ ▄ ▗ ▄ ▗ ▄ "
715 PRINT"";
720 PRINT" ";:FORK=1TO38:PRINT"";
723 NEXT:PRINT
725 FORK=1TO5:PRINTA$A$LEFT$(A$,19):NEXT
730 FORJ=1TO4
735 PRINT"|";:FORK=1TO19:PRINT"  |";:NEXT
740 PRINT:NEXT
745 PRINT"";:FORK=1TO18:PRINT"";
746 NEXT:PRINT""
750 GOTO750
```

ITALIA: abbiamo preparato tre esempi, i primi due, ES43 e ES44, in bassissima risoluzione, il secondo, ES45, in bassa risoluzione. Potranno servirti di spunto per disegnare sullo schermo.

IN BASSISSIMA RISOLUZIONE: si possono scegliere diverse strade:

- usare una serie di PRINT con le stringhe alfanumeriche già preparate,
- usare una serie di PRINT che leggano da un file interno, DATA, le stringhe da stampare,
- codificare in qualche modo il disegno.

Noi abbiamo scelto l'ultima strada poichè il nostro disegno richiede solo due differenti caratteri. Stabilito ciò, abbiamo analizzato il disegno che volevamo

ottenere e abbiamo definito una serie di stringhe numeriche, ciascuna delle quali descrive una linea in questo modo:

XX1 YY1 XX2 YY2...

dove:

XXi è il numero di caratteri di tipo A,
YYi è il numero di caratteri di tipo B.

Noi abbiamo usato come carattere A lo spazio e come carattere B lo spazio inverso, ma si può usare una coppia diversa di caratteri, apportando le necessarie modifiche.

```
1 REM ES43
5 REM ITALIA
10 RESTORE:READNL:PRINT"■";
20 FORC=1TONL:READA$
30 FORI=1TOLEN(A$)STEP4
40 A=VAL(MID$(A$,I,2)):PRINTSPC(A);
50 A=VAL(MID$(A$,I+2,2))
55 FORK=1TOA:PRINT"■ ■";:NEXT
60 NEXT:IF POS(0)<39AND C<NL THENPRINT
70 NEXT
80 PRINT"■"SPC(180)"ITALIA";
90 GOTO90
1000 DATA25,0903,0706,03010109,030101080101
1005 DATA0110,0210,0111,0112,03020307
1010 DATA0808,0909,08010112,1209,1312
1015 DATA06011010,060209060301,05031203
1020 DATA06021403,05031502,05031501,06011601
1025 DATA21010101,1606,1605,1903
```

Una modifica può essere usare dei caratteri "O" invece degli spazi e attribuire i giusti colori a terra e mare, come segue:

```
1 REM ES44
5 REM ITALIA
10 RESTORE:READNL:PRINT"■";
13 POKE53280,1:POKE53281,1
15 PRINT"■";
20 FORC=1TONL:READA$
```

```
30 FORI=1TOLEN(A$)STEP4
40 A=VAL(MID$(A$,I,2))
43 FORK=1TOA:PRINT"●";:NEXT
45 PRINT"▗";
50 A=VAL(MID$(A$,I+2,2))
53 FORK=1TOA:PRINT"▘ ▝";:NEXT
55 PRINT"▖";
60 NEXT:IFPOS(0)<39THEN100
65 IF C<NLTHENPRINT
70 NEXT
80 PRINT"▚"SPC(180)"▚ITALIA";
90 GOTO90
100 FORK=POS(0)TO28:PRINT"●";:NEXT:GOTO65
1000 DATA25,0903,0706,03010109,030101080101
1005 DATA0110,0210,0111,0112,03020307
1010 DATA0808,0909,08010112,1209,1312
1015 DATA06011010,060209060301,05031203
1020 DATA06021403,05031502,05031501
1025 DATA06011601,21010101,1606,1605,1903
```

IN BASSA RISOLUZIONE: questa volta abbiamo usato un diverso sistema di codifica, associando ai 16 possibili caratteri della bassa risoluzione altrettante lettere dell'alfabeto, come risulta dalla Figura 2.12.

CORRISPONDENZA LETTERE/CARATTERI

A	SPAZIO	B	▪	C	▪	D	▬
E	▪	F	▮	G	▚	H	▙
I	▪	J	▞	K	▮	L	▟
M	▬	N	▛	O	▜	P	■

Figura 2.12 Esempio di codifica di caratteri per GRAFICA a BASSA RISOLUZIONE

Ogni linea è stata codificata come:

XX,YYYYYY,XX,YYY,...,-1

dove:

XX indica il numero degli spazi,
YY..YY indica le lettere da usare e quindi i caratteri grafici,
il numero -1 chiude la linea.

Per diminuire la possibilità di errori abbiamo usato un'istruzione DATA per ogni linea da disegnare.

```
1 REM ES45
3 REM ITALIA
5 RESTORE:DIMA$(15)
6 FORK=0TO15:READA$(K):NEXT
10 READNL:PRINT"[illegible]";
15 POKE53280,0:POKE53281,0:POKE646,7
20 FORK=1TONL:PRINTSPC(5);
30 FORJ=0TO1:J=0:READA
40 IFA=-1THENJ=1:NEXT:GOTO90
50 PRINTSPC(A);
60 READA$
70 FORI=1TOLEN(A$):A=ASC(MID$(A$,I,1))-65
75 PRINTA$(A);:NEXT
80 NEXT
90 PRINT:NEXT:PRINT"[illegible]"SPC(180)"ITALIA";
95 GOTO95
100 DATA" ","[illegible]","[illegible]","[illegible]","[illegible]","[illegible]","[illegible]","[illegible]"
105 DATA"[illegible]","[illegible]","[illegible]","[illegible]","[illegible]"
110 DATA"[illegible]","[illegible]","[illegible]"
1000 REM DATA PER <<ITALIA>>
1001 DATA22,6,CL,-1
1002 DATA5,LLPHD,-1
1003 DATA2,HKLLPPPH,-1
1004 DATA0,KPPLPPPPPMF,-1
1005 DATA0,LPPPPPPP,-1
1006 DATA0,OPPPPPPPE,-1
1007 DATA0,OPNMPPPPH,-1
1008 DATA1,J,2,EPPPPD,-1
1009 DATA5,KPPPPF,-1
1010 DATA5,JPPPPH,-1
```

```
1011 DATA6,IOPPPH,-1
1012 DATA7,IPPPPPPE,-1
1013 DATA8,IOPPPPHD,-1
1014 DATA2,BLB,6,OPPPPPD,-1
1015 DATA2,OPF,6,IMPPNIOF,-1
1016 DATA2,KPF,8,IPF,2,E,-1
1017 DATA2,LPF,9,KPB,-1
1018 DATA2,ONE,10,NE,-1
1019 DATA14,KF,-1
1020 DATA9,CDDDPI,-1
1021 DATA9,OPPPF,-1
1022 DATA10,IMPP,-1
```

MARYLIN: ecco, in ES46, le linee DATA per realizzare su 25 linee un ritratto, un po' allungato, di MARYLIN MONROE.

```
 1 REM ES46
 3 REM MARYLIN
 5 RESTORE:DIMA$(15)
 6 FORK=0TO15:READA$(K):NEXT
 10 READNL:PRINT"[illegible]";
 15 POKE53280,0:POKE53281,0:POKE646,8
 20 FORK=1TONL:PRINTTAB(3);
 30 FORJ=0TO1:J=0:READA
 40 IFA=-1THENJ=1:NEXT:GOTO90
 50 PRINTSPC(A);
 60 READA$
 70 FORI=1TOLEN(A$):A=ASC(MID$(A$,I,1))-65
 75 PRINTA$(A);:NEXT
 80 NEXT
 90 IFPOS(0)<39ANDK<NLTHENPRINT
 95 NEXT:PRINT"[illegible]"SPC(31)"[illegible]MARYLIN";
 98 GOTO98
 100 DATA" ","[illegible]","[illegible]","[illegible]","[illegible]","[illegible]","[illegible]","[illegible]"
 105 DATA"[illegible]","[illegible]","[illegible]","[illegible]","[illegible]"
 106 DATA"[illegible]","[illegible]","[illegible]"
 999 REM DATA PER <<MARYLIN>>
 1000 DATA25,1,PPPPEADPPPPPPPPPPPPPPPPPPPH,-1
 1001 DATA0,KPPPPALPPPPPPPPPPPPPPPPPPPPB,-1
```

```
1002 DATA0,KPPPPKPPPPPPPPPPPPNMPPPPPPPH,-1
1003 DATA1,PNBPPPPPPPPPPPEDDDOPPPPOPPB,-1
1004 DATA1,IPHIPPPPPPPPNDPPPPPPPPEKPPH,-1
1005 DATA1,EPPFOPPOPPPPPPPPPPPPPPPAAIOP,-1
1006 DATA1,IIPPKPNEKPPPPPNMMNEPPPADBAO,-1
1007 DATA2,EOPAEDNMMPPPNDDDLPPPPLPPHAPB,-1
1008 DATA2,CJPACPPPHOPPPPPPPPPPPPPPPBPF,-1
1009 DATA2,IAFAKPNELKPPPPPPPPPPPPPPPPPP,-1
1010 DATA3,LFAIACLPPPPPPPPPPPPPPPPPPPP,-1
1011 DATA3,IHBACPPPKPPPPPPPPPPPPPPPPPO,-1
1012 DATA3,KPFAKPPFIPNDLPPHOPPPKPPPN,-1
1013 DATA3,LNPAPPPHLPPPPPPPHOPPKPPPECF,-1
1014 DATA3,OPIAOPPPPPPNNMPPPPPPNOPNALF,-1
1015 DATA3,IPFAAOPPONACDDBPPPPPPFAAAPP,-1
1016 DATA4,IPBAAOPKALPMCBPPPPPPFACPPPB,-1
1017 DATA3,CAPPACKPNAMCLNCPPPPPPFAAALPF,-1
1018 DATA3,KKPPBABPPBCPEBLPPPPPPFADCPPF,-1
1019 DATA3,BAIOHACIPHAAILPPPPHPPFAPPPNE,-1
1020 DATA3,MAAAOBAHOPHDPPPPPNLPPFAKPNHDDD,-1
1021 DATA4,FAAAAAIFOPPPPPPPKPPPFKPNLPPPPPD,-1
1022 DATA4,IAAAAAAICPPPPPPELPPPFLNCPPPPPPP,-1
1023 DATA5,GAAAALDPHJOPNEALPPPPFMCPPPPPPPP,-1
1024 DATA6,GALPPPPPPHAAKPPPPPFALPPPPPPPP,-1
```

CARTE DA GIOCO: dato che tra i caratteri grafici sono compresi i 4 semi delle carte francesi, abbiamo preparato una routine per disegnare le carte, ES47.

```
1 REM ES47
5 SE$="♥♦♠♣":N$="*A234567891JQK"
8 POKE53281,1
10 C$="▚":C1$=" |     | ":A$=" |  ":B$="  | "
15 CC=646:POKECC,6
18 SP$="▌▌▌
```

```
50 GETA$:IFA$=""THEN50
55 RUN
3000 PRINT"┌──────────┐"SPC(30);
3005 FORK=0TO11:PRINT"│▒▒▒▒▒▒▒▒▒▒│"SPC(30);
3010 NEXT
3015 PRINT"└──────────┘";
3020 PRINT"▗▄▄▄
```

```
3205 FORK=1TO3:PRINT" | "D$" "D$" | "SP$;
3210 PRINTC1$SP$;:NEXT:PRINTC1$SP$;
3215 RETURN
3220 PRINTC1$SP$C1$SP$;
3225 PRINT" | "D$" "D$" | "SP$;
3230 PRINTA$D$B$SP$;
3235 FORK=1TO2:PRINT" | "D$" "D$" | "SP$;
3240 PRINTC1$SP$;:NEXT
3245 PRINTC1$SP$;:RETURN
3250 PRINTC1$SP$;
3255 FORK=1TO4:PRINT" | "D$" "D$" | "SP$;
3260 PRINTC1$SP$;:NEXT:RETURN
3265 RETURN
3270 PRINTC1$SP$" | "D$" "D$" | "SP$;
3275 PRINTA$D$B$SP$;
3280 FORK=1TO3:PRINT" | "D$" "D$" | "SP$C1$;
3285 PRINTSP$;:NEXT:RETURN
3290 PRINTC1$SP$;
3295 FORK=1TO2:PRINT" | "D$" "D$" | ";
3300 PRINTSP$A$D$B$SP$;:NEXT
3305 FORK=1TO2:PRINT" | "D$" "D$" | "SP$C1$SP$;
3310 NEXT:RETURN
3320 PRINT" | ◥◪ ▬◤ ⅄ "SP$;
3321 PRINT" | "◪▔▬ || "SP$;
3322 PRINT" | ◪◤"D$"▬◤◪|▬ "SP$;
3323 PRINT" |◪◤"D$"/—▬ "SP$;
3324 PRINT" |◪"D$"—+—"D$"▬ "SP$;
3325 PRINT" |◪⌐—/"D$"▬◥ "SP$;
3326 PRINT" |◪|◤"D$"▬◤ | "SP$;
3327 PRINT" ||.◪▙ | "SP$;
3328 PRINT" ⊢◪◤ ◥▬ | "SP$;
3329 RETURN
3330 PRINT" | ◥◪ ▬◤ ⅄ "SP$;
3331 PRINT" | ◪◤O◥▬H "SP$;
3332 PRINT" |◪◤ ▬◤◪|▬ "SP$;
3333 PRINT" |◪\"D$"\\◤▬ "SP$;
3334 PRINT" |◪\\"D$"\\▬ "SP$;
3335 PRINT" |◪/\\"D$"\▬ "SP$;
3336 PRINT" |◪|◤ ▟◤ "SP$;
3337 PRINT" H◥◪O▬◤ | "SP$;
```

```
3338 PRINT" ▛▜ ▜▟ ▌"SP$;
3339 RETURN
3340 PRINT" ▌▚▞ ▚▟▌▌"SP$;
3341 PRINT" ▌ ▚▞▜▟▛▌"SP$;
3342 PRINT" ▛▜ "D$"▜▞▞▌▟ "SP$;
3343 PRINT" ▛▞ "D$"/▁▟ "SP$;
3344 PRINT" ▛▞"D$"/+/"D$"▟ "SP$;
3345 PRINT" ▛▞▔/"D$" ▟ "SP$;
3346 PRINT" ▛▞▌▜"D$" ▟▜▌ "SP$;
3347 PRINT" ▛▞▚▞▟ ▌"SP$;
3348 PRINT" ▌▌▞▚ ▟▚ ▌"SP$;
3349 RETURN
3350 PRINTC1$SP$" ▌****▌"SP$;
3351 PRINT" ▌  ** ▌"SP$;
3352 PRINT" ▌  ** ▌"SP$;
3353 PRINT" ▛▜ ** ▌"SP$;
3354 PRINT" ▛▜ ** ▌"SP$;
3355 PRINT" ▛▜ ** ▌"SP$;
3356 PRINT" ▌**  ▌"SP$C1$SP$;:RETURN
```

CIFRE GRANDI: seguono due routine per ingrandire le cifre 2*3 e 5*4.

Ingrandimento 2*3 con il programma ES48 : vogliamo generare cifre come quelle riportate nella Figura 2.13, per scrivere un numero intero N.

Figura 2.13 Ingrandimenti cifre 2X3

Come per le carte da gioco possiamo usare 10 piccole routine che disegnano ognuna delle 10 cifre.

```
1 REM ES48
5 REM CIFRE 2X3
10 INPUT"NUMERO: ";N
```

```
20 GOSUB200
99 GOTO99
200 N$=MID$(STR$(N),2,LEN(STR$(N))-1)
205 G$="▌▌▌"
210 FORK=1TOLEN(N$):A=VAL(MID$(N$,K,1))+1
220 ONAGOSUB250,252,254,256,258
221 IFA<=5THEN225
223 ON(A-5)GOSUB260,262,264,266,268
225 PRINT"▝▘";:NEXT:RETURN
250 PRINT"▗▄▖ "G$"▌▌ "G$"▗▄▖ ";:RETURN
252 PRINT"▗▝▖ "G$"▗▝▖ "G$" ▝ ";:RETURN
254 PRINT"▗▄▖ "G$"▗▟▖ "G$"▗▄▖ ";:RETURN
256 PRINT"▗▄▖ "G$" ▜ "G$"▗▄▖ ";:RETURN
258 PRINT"▌ "G$"▗▜▖ "G$" ▝ ";:RETURN
260 PRINT"▗▟▖ "G$"▗▄▖ "G$"▗▄▖ ";:RETURN
262 PRINT"▗▟▖ "G$"▗▟▖ "G$"▗▄▖ ";:RETURN
264 PRINT"▗▄▖ "G$" ▌ "G$" ▝ ";:RETURN
266 PRINT"▗▟▖ "G$"▗▟▖ "G$"▗▄▖ ";:RETURN
268 PRINT"▗▟▖ "G$"▗▄▖ "G$" ▝ ";:RETURN
```

Ingrandimento 5*4 con il programma ES49: possiamo ingrandire ancora le cifre precedenti, usando un sistema di codifica procediamo così: abbiamo osservato che per ogni riga le possibili combinazioni di punti sono quelle riportate nella Figura 2.14.

COMBINAZIONI POSSIBILI PER OGNI RIGA

1 2 3 4 5 6

CIFRA 1

1
3
1
1
6

Figura 2.14 Combinazioni di punti per ingrandire cifre 5X4

e che ogni cifra può essere descritta con 5 numeri che corrispondono alle combinazioni di punti da utilizzare. Nella Figura 2.15 abbiamo rappresentato tutte le cifre ingrandite.

Figura 2.15 Ingrandimento cifre 5X4

```
1 REM ES49
5 REM CIFRE 5X4
10 DIMA$(6):DIMB$(9)
20 FORK=0TO6:READA$(K):NEXTK
30 FORK=0TO9:READB$(K):NEXTK
33 DATA"  ▗▄▖","  ▗▄▖ ","  ▗▄  ","▗▄▖  "
34 DATA"▗▄▖ ▗▄▖","▗▄  ","▗▄  "
35 DATA65556,27226,61646,61316,44561,64616
36 DATA64656,61111,65656,65611
50 INPUT"NUMERO: ";N
60 GOSUB200
99 GOTO99
200 N$=MID$(STR$(N),2,LEN(STR$(N))-1)
205 G$="▚▞▞▞"
210 FORK=1TOLEN(N$):A=VAL(MID$(N$,K,1))
220 FORI=1TO5:B=VAL(MID$(B$(A),I,1))-1
230 PRINTA$(B)"▚▞▞▞";
240 NEXTI
250 PRINT"▝▜▜▜▛▟▟▟▟";
260 NEXTK
270 RETURN
```

LETTERE E NUMERI: usando i caratteri grafici possiamo ingrandire lettere e numeri. Nei due esempi precedenti abbiamo usato due metodi diversi di ingrandimento; un metodo può essere quello di leggere in ROM le descrizioni dei caratteri e

poi sostituire a ogni puntino un carattere grafico adatto (sia in bassissima che in bassa risoluzione). Altro metodo può essere quello di preparare per ogni carattere una stringa che lo descrive, oppure, fissate le dimensioni volute, preparare delle stringhe che contengano ognuna una parte di tutti i caratteri, nelle quali si possono ritrovare le sottostringhe che compongono il carattere ingrandito che serve.
Ad esempio per l'alfabeto riportato in Figura 2.16, possiamo definire 8 stringhe, 4 per le lettere maiuscole e 4 per le minuscole, nelle quali ricercare, usando il codice ASCII di ogni lettera come indice, per trovare le 4 sottostringhe che lo descrivono. Segue come esempio il programma ES50.

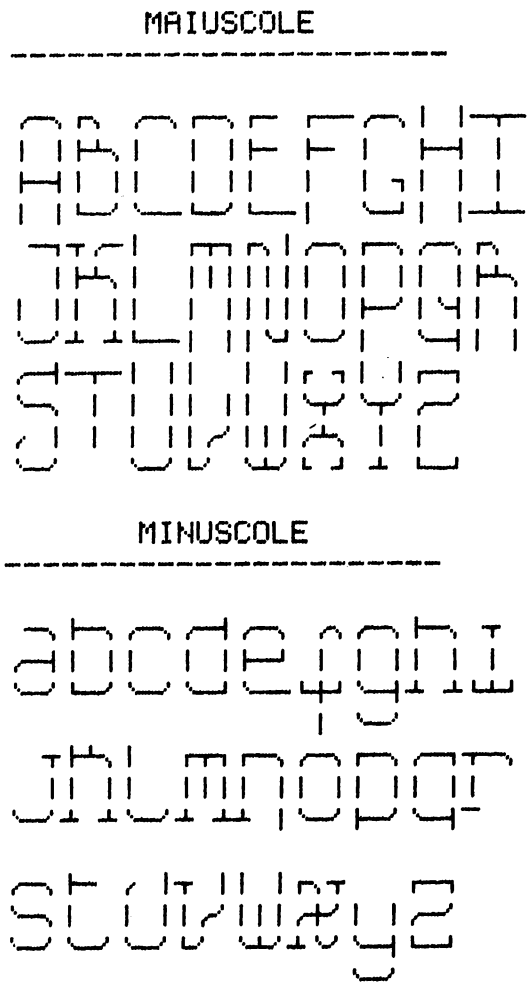


Figura 2.16 Alfabeto

```
1 REM ES50
5 REM ALFABETO
10 DIMA$(3),B$(3),C$(3)
11 Q$="▮▮▮▮▯▯"：W$="▮▮▮▮▯▯▯▯"
13 FORJ=1TO3：FORK=0TO3：READD$：A$(K)=A$(K)+D$
15 NEXT：NEXT
18 FORJ=1TO3：FORK=0TO3：READD$：B$(K)=B$(K)+D$
20 NEXT：NEXT
28 PRINTCHR$(14);
30 INPUT"▯STRINGA：";S$：S$=LEFT$(S$,13)
35 PRINT"▯▯▯"CHR$(142);
40 FORI=1TOLEN(S$)：CH=ASC(MID$(S$,I,1))
45 GOSUB300
50 FORJ=0TO3：PRINTC$(J)Q$;：NEXT：PRINTW$;：NEXTI
55 GETA$：IFA$<>"▮"THEN55
60 GOTO28
300 REM SCRIVE CARATTERE
320 IFCH<65THEN380
325 IFCH>90THEN340
330 CH=CH-65
335 FORK=0TO3：C$(K)=MID$(B$(K),CH*3+1,3)：NEXT
338 RETURN
340 IFCH<193THEN380
345 IFCH>218THEN380
350 CH=CH-193
355 FORK=0TO3：C$(K)=MID$(A$(K),CH*3+1,3)：NEXT
358 RETURN
380 FORK=0TO3：C$(K)="   "：NEXT：RETURN
1000 REM MAIUSCOLE
1001 DATA"▗▛▚ ▗▛▜▗▛▜ ▜▛▘▌ ▌▀▛"
1002 DATA"▌ ▌▙▗▌ ▌ ▌▙▖ ▙▖ ▌ ▗▙▖ ▌ "
1003 DATA"▙▖▌ ▌▌ ▌ ▌▌ ▌ ▗▖▌ ▌ ▌ "
1004 DATA"▝▀▘▝▀▘▝▀▘▝▀▘ ▝▀▘ ▝▀▘▝▀▘"
1005 DATA" ▀▜▛▗▌ ▛▜▛▗▌▛▜▛▜▛▜▛▜ "
1006 DATA"  ▌▙▗▌ ▌▌▌▌▌▌▌ ▌▌ ▌▌ ▌▙▗"
1007 DATA"▌ ▌▌ ▌▌ ▌ ▌▌▌▌▌ ▌▙▛▌▚▌ ▌"
1008 DATA"▝▀▘▝▀▘▝▀▘▝▀▘ ▌▌▝▀▘ ▝▀▘ ▝▀▘ ▌"
1009 DATA"▗▛▀▜▛▀▜ ▌▌  ▌ ▌▛▜▌ ▌▛▜"
1010 DATA"▚ ▌ ▌ ▌▌ ▌▌ ▌▛▚▛▚▗▛"
1011 DATA"▗▘▌ ▌ ▌ ▌▌▗▘▌▌▌▛▚ ▌ ▌ "
1012 DATA"▝▀  ▝▀▘▝▀▘▝▀▘▝▀▘▝▀▘▝▀▘"
```

```
1020 REM MINUSCOLE
1021 DATA"
1022 DATA"
1023 DATA"
1024 DATA"
1025 DATA"
1026 DATA"
1027 DATA"
1028 DATA"
1029 DATA"
1030 DATA"
1031 DATA"
1032 DATA"
```

Per poter stampare anche i numeri, bisogna prima preparare la descrizione riportata in Figura 2.17, quindi modificare il programma, come indicato in ES51, che non viene listato, ma è registrato sulla cassetta.

NUMERI

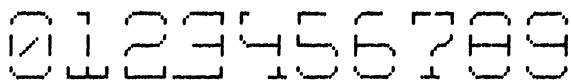


Figura 2.17 Cifre numeriche

Prova a modificare il programma per ottenere anche il carattere "@", oppure per ottenere altri caratteri speciali.

MESSAGGI SUL VIDEO: vogliamo realizzare scritte colorate che scorrono sul video. Ti proponiamo il programma ES52; in esso scriviamo i primi 40 caratteri della stringa e poi spostiamo il primo carattere in fondo alla stringa, metodo che risulta abbastanza veloce.

```
1 REM ES52
3 REM PAROLE CHE SCORRONO
5 S$="":FORK=1TO40:S$=S$+" ":NEXTK
10 INPUT"STRINGA1";A$:A$=S$+A$
20 INPUT"STRINGA2";B$:A$=A$+" "+B$
30 PRINT"[illegible]";
40 PRINTLEFT$(A$,40)"[illegible]";:FORK=1TO100:NEXT
50 A$=MID$(A$,2)+LEFT$(A$,1):GOTO40
```

Puoi proporti di ottenere lo scorrimento in altre direzioni; buon lavoro!

OROLOGI: dato che il calcolatore misura il tempo trascorso dall'accensione usando 3 byte consecutivi in pagina 0, in 60-esimi di secondo, e che si può intervenire a modificare il valore iniziale dei byte, è possibile gestire un orologio. Per accedere da BASIC all'orologio interno del COMMODORE 64 sono disponibili le funzioni TI e TI$.
Segue il programma ES54 che visualizza l'orologio DIGITALE.

```
5 REM ES54
10 INPUT"[illegible]ORA (HHMMSS)";A$
15 IFLEN(A$)<>6THEN10
17 FORK=1TO6
19 IFMID$(A$,K,1)<"0"ORMID$(A$,K,1)>"9"THEN10
21 X1$=MID$(A$,1,2)
23 X2$=MID$(A$,3,2)
25 X3$=MID$(A$,5,2)
27 IFX1$>"23"ORX2$>"59"ORX3$>"59"THEN10
28 TI$=A$:PRINT"[illegible]";
30 FORT=0TO1:GOSUB100:T=0:NEXT
100 REM ORA CON [PRINT]
105 A$=TI$
110 PRINT"[illegible]"LEFT$(A$,2)":";
113 PRINTMID$(A$,3,2)":"RIGHT$(A$,2);
115 RETURN
```

2.7 COLLOQUIO SULLO SCHERMO

La parte fondamentale di un programma che non sia A SE STANTE è la interazione con l'utente. In alcuni programmi, specialmente nei videogiochi, questa interazione è resa il più semplice possibile, si preme un tasto, si sposta una levetta o una manopola e subito si può vedere sullo schermo l'effetto della propria azione. Questa è anche l'attuale tendenza negli ultimi modelli di calcolatori e nei più recenti programmi prodotti: l'utente deve ricordare il meno possibile; deve essere il programma a fornire una guida continua in modo che le operazioni siano semplici e prevedibili. Questo scopo viene raggiunto usando sofisticate tecniche di programmazione e spesso ricorrendo alla grafica raffinata. Qui ci limitiamo a vedere qualche esempio di interazione calcolatore-programma-utente ottenuta in MODO CARATTERI.

2.7.1 Il linguaggio naturale - I sottolinguaggi

Molto spesso, quando la velocità non è essenziale, si può ricorrere alla tastiera come principale dispositivo di ingresso dati (l'uscita è sempre sottinteso sia sul video) e i sistemi per garantire l'interattività sono di tipo conversazionale o guidati da menù. Nel primo caso si possono fare vari esempi. Lo stesso BASIC è di tipo conversazionale, anche se usa come possibili comandi solo un sottoinsieme dei verbi della lingua inglese.
Sarebbe sicuramente comodo poter impartire comandi come: CERCA IL VALORE MASSIMO DELLA FUNZIONE:..... TRA 0 e 10, oppure: CANCELLA LA REGISTRAZIONE DEL SIGNOR BIANCHI BRUNO. O anche, per risparmiare un pò sulla interpretazione, usare un linguaggio più telegrafico. Un esempio ben riuscito di questo tipo di interazione sono i cosiddetti ADVENTURE-GAME (giochi di avventura), nei quali il programma descrive una situazione in cui il giocatore si trova ed egli può dire al programma come decide di comportarsi, utilizzando un vocabolario alquanto ricco (purtroppo spesso solo in inglese, lingua che risulta molto adatta, dato che si basa su regole sintattiche abbastanza semplici per essere programmate).
Nella maggior parte degli altri casi il vocabolario viene ridotto ai soli verbi, seguiti da parametri numerici o alfabetici separati da barre, virgole o spazi, o altri caratteri separatori. Abbiamo già citato il BASIC, ma anche altri programmi di utilità generale usano queste tecniche.
Il più SCOMODO per l'utente, ma più semplice da programmare e più veloce da usare, è il sistema del riconoscimento della lettera INIZIALE, che fa riconoscere i comandi grazie al fatto che per distinguerli bastano la prima o le prime due lettere del verbo che li descrive (CA per CANCELLA e CE per CERCA). Anche se all'inizio può creare qualche perplessità, si impara rapidamente a usare questo metodo per fornire ordini ai programmi.

Inoltre, come ulteriore semplificazione, può essere preparata una specie di maschera da sovrapporre alla tastiera, che rechi un memo dei comandi. Alcuni programmi mostrano tale memo perennemente in una sezione del video.
Un sistema analogo è quello DOMANDA/RISPOSTA, utilizzabile quando il programma abbia un utilizzo specifico (operazioni sulle periferiche, procedure di calcolo,...); in questo caso il programma prende l'iniziativa cominciando a porre domande, a cui l'utente deve rispondere spesso solo con una parola o un numero in base al quale il programma continua a chiedere i successivi elementi di cui ha bisogno per poter compiere una determinata operazione.

2.7.2 I menù

E' un altro sistema diffusissimo per proporre delle scelte; esso consiste in un elenco delle operazioni possibili.

Un modo di presentare un menù è l'elencazione delle operazioni possibili una sotto l'altra, con a fianco il tasto da premere per la selezione. Basta premere il tasto giusto e il programma procede. Ovviamente non viene accettato un tasto non congruente. Segue il programma esempio ES55.

```
1 REM ES55
5 REM SCELTA PER ELENCAZIONE
6 PRINT"[illegible]"TAB(16)"M E N U'":PRINT
10 FORK=1TO5:READA$:PRINTTAB(10)A$:NEXTK
15 PRINT:PRINTTAB(10)"SCELTA >";
20 POKE198,0
30 GETA$:IFA$<"1"ORA$>"5"THEN30
40 PRINTA$;
50 POKE198,0
60 GETR$:IFR$=""THEN60
70 IFR$<>CHR$(13)THENPRINT"[illegible]":GOTO30
75 PRINT
80 ONVAL(A$)GOTO100,200,300,400,500
100 PRINT"SCELTA N.1 ":STOP
200 PRINT"SCELTA N.2 ":STOP
300 PRINT"SCELTA N.3 ":STOP
400 PRINT"SCELTA N.4 ":STOP
500 PRINT"SCELTA N.5 ":STOP
1000 DATA"-OPERAZIONE 1 - 1 -"
```

```
1001 DATA"-OPERAZIONE 2 - 2 -"
1002 DATA"-OPERAZIONE 3 - 3 -"
1003 DATA"-OPERAZIONE 4 - 4 -"
1004 DATA"-OPERAZIONE 5 - 5 -"
```

Un altro modo è la presentazione di un elenco, con la possibilità di muoversi con un CURSORE, rappresentato da un particolare carattere, per puntare all'operazione desiderata. Trattiamo questo nel programma ES56.

```
0 REM ES56
3 S$="                                                  "
5 PRINT"[illegible]"SPC(136)"M E N U'"SPC(32)"========="
10 DIMA$(5):FORK=1TO5:READA$(K):NEXT
15 PRINT:POKE53281,1
20 FORK=1TO5:PRINTTAB(14)A$(K):PRINT:NEXT
30 GOSUB100
40 ONSGOTO50,60,70,80,90
45 STOP
50 PRINT"SCELTA N. 1":STOP
60 PRINT"SCELTA N. 2":STOP
70 PRINT"SCELTA N. 3":STOP
80 PRINT"SCELTA N. 4":STOP
90 PRINT"SCELTA N. 5":STOP
95 DATA" 1 PRIMO   "," 2 SECONDO "
96 DATA" 3 TERZO   "
97 DATA" 4 QUARTO  "," 5 QUINTO  "
100 REM SCEGLIE
105 PRINT"[illegible]"SPC(11)"[F1] SPOSTA IN GIU'";
110 PRINTSPC(21)"[RETURN] CONFERMA"
115 S=1
120 PRINT"[illegible]"SPC(120):FORK=1TOS
123 PRINTSPC(80);:NEXT
125 PRINTSPC(13)" [illegible] "SPC(27);
130 PRINT"[illegible] "A$(S)"[illegible] "SPC(27);
135 PRINT" "[illegible] "SPC(27)"[illegible]";
140 GETQ$:IFQ$<>CHR$(13)ANDQ$<>"[illegible]"THEN140
145 PRINTS$" "A$(S)S$" ";
150 IFQ$="[illegible]"THENS=S+1+5*(S=5):GOTO120
155 PRINT"[illegible]"S$S$;
160 PRINT"[illegible]"SPC(255)SPC(255)SPC(130);:RETURN
```

In certi casi tale CURSORE può essere rappresentato anche da un disegno abbastanza complesso che occupi più posizioni carattere. Molto bello è il modo di selezione presentato dal programma MAGIC DESK, nel quale si sposta una manina con l'indice puntato per mezzo di un Joystick.
Quello che è essenziale è che la selezione sia sicura e non possa dar luogo ad ambiguità di interpretazione.

Alcuni programmi abbastanza sofisticati consentono l'uso di un particolare tasto di HELP per richiamare un menù sul video in caso di bisogno, senza ovviamente perdere il precedente contenuto, che viene ripristinato quando non serve più il menù.

2.7.3 Come evidenziare una scelta

Una volta che l'utente ha operato una scelta, è bene che il programma lo informi che ha ricevuto, e, nei casi più delicati, chieda conferma. Il programma può evidenziare la scelta attivando il campo inverso, realizzando una sottolineatura, modificando il colore di una zona o altro.
L'importante è che l'utente si possa rendere conto di avere o non avere dato al programma l'ordine desiderato. Segue come esempio il programma ES57.

```
0 REM ES57
3 S$="                                                    "
5 PRINT"■"SPC(96)"M E N U'"SPC(32)"========="
10 DIMA$(5):FORK=1TO5:READA$(K):NEXT
15 PRINT:POKE53281,1
20 FORK=1TO5:PRINTTAB(14)A$(K):PRINT:NEXT
30 GOSUB100
40 ONSGOTO50,60,70,80,90
45 STOP
50 PRINT"SCELTA N.  1":STOP
60 PRINT"SCELTA N.  2":STOP
70 PRINT"SCELTA N.  3":STOP
80 PRINT"SCELTA N.  4":STOP
90 PRINT"SCELTA N.  5":STOP
95 DATA" 1 PRIMO    "," 2 SECONDO "
96 DATA" 3 TERZO    "
97 DATA" 4 QUARTO   "," 5 QUINTO  "
100 REM SCEGLIE
```

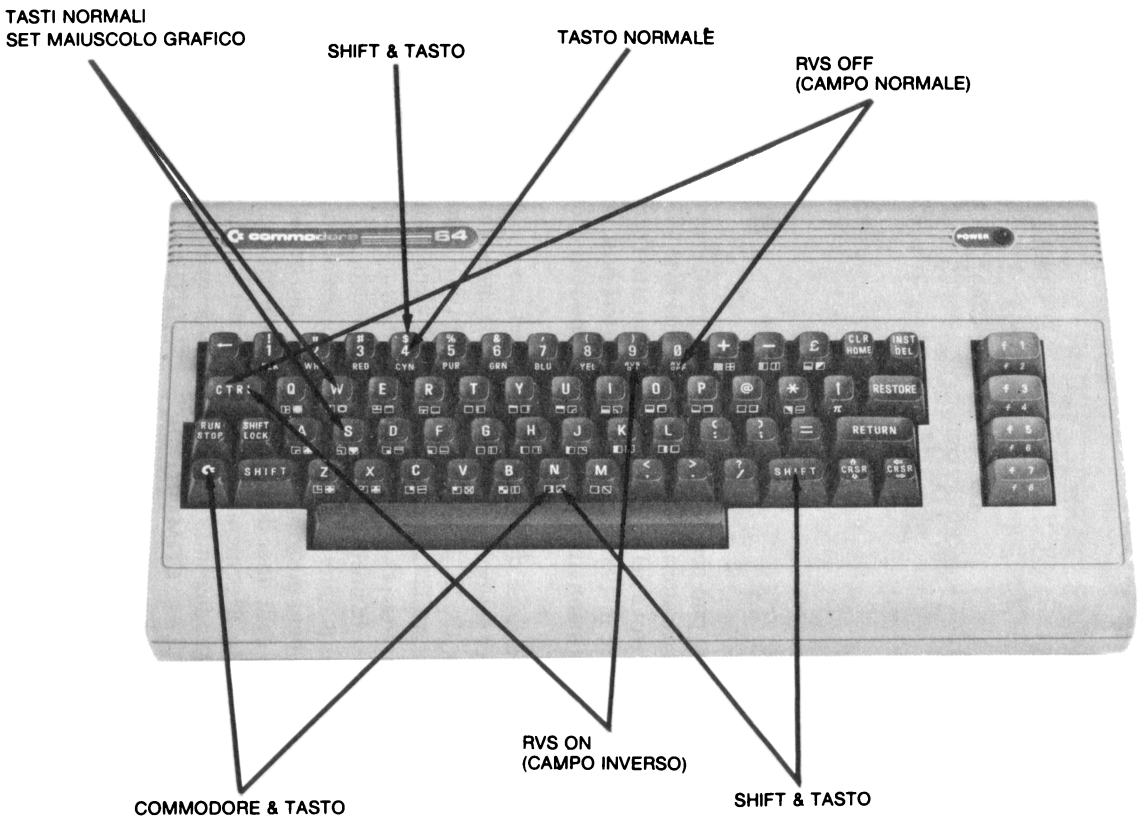
```
105 PRINT"▒"SPC(255)SPC(255)SPC(50)"SCELTA >";
110 GETQ$:IFQ$<"1"ORQ$>"5"THEN110
115 PRINTQ$"▒";:FORK=0TOVAL(Q$):PRINT"▒▒";:NEXT
120 PRINTSPC(54)"▒"A$(VAL(Q$))"▒";
125 PRINT"▒"SPC(10)"▒▒[ RETURN --> CONFERMA]";
130 PRINTSPC(18)"▒[ SPAZIO --> ANNULLA ]▒▒"
135 POKE198,0:S=VAL(Q$)
140 GETQ$:IFQ$<>" "ANDQ$<>CHR$(13)THEN140
145 PRINT"▒"S$S$SPC(255)SPC(225);
150 IFQ$=CHR$(13)THENPRINTS$:RETURN
155 PRINTSPC(8)" ";
160 PRINT"▒";:FORK=0TOS:PRINT"▒▒";:NEXT
165 PRINTSPC(54)A$(S);:GOTO105
```

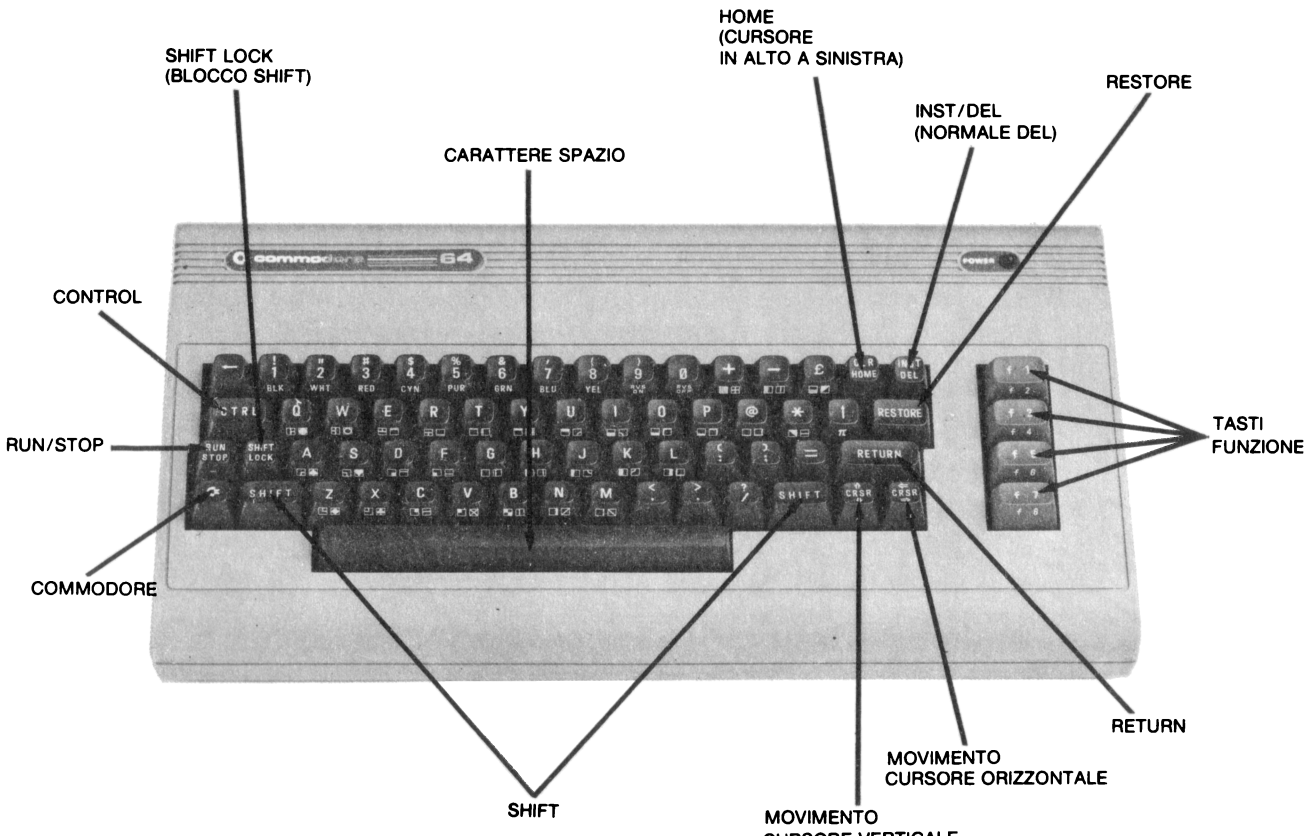
CAPITOLO 3

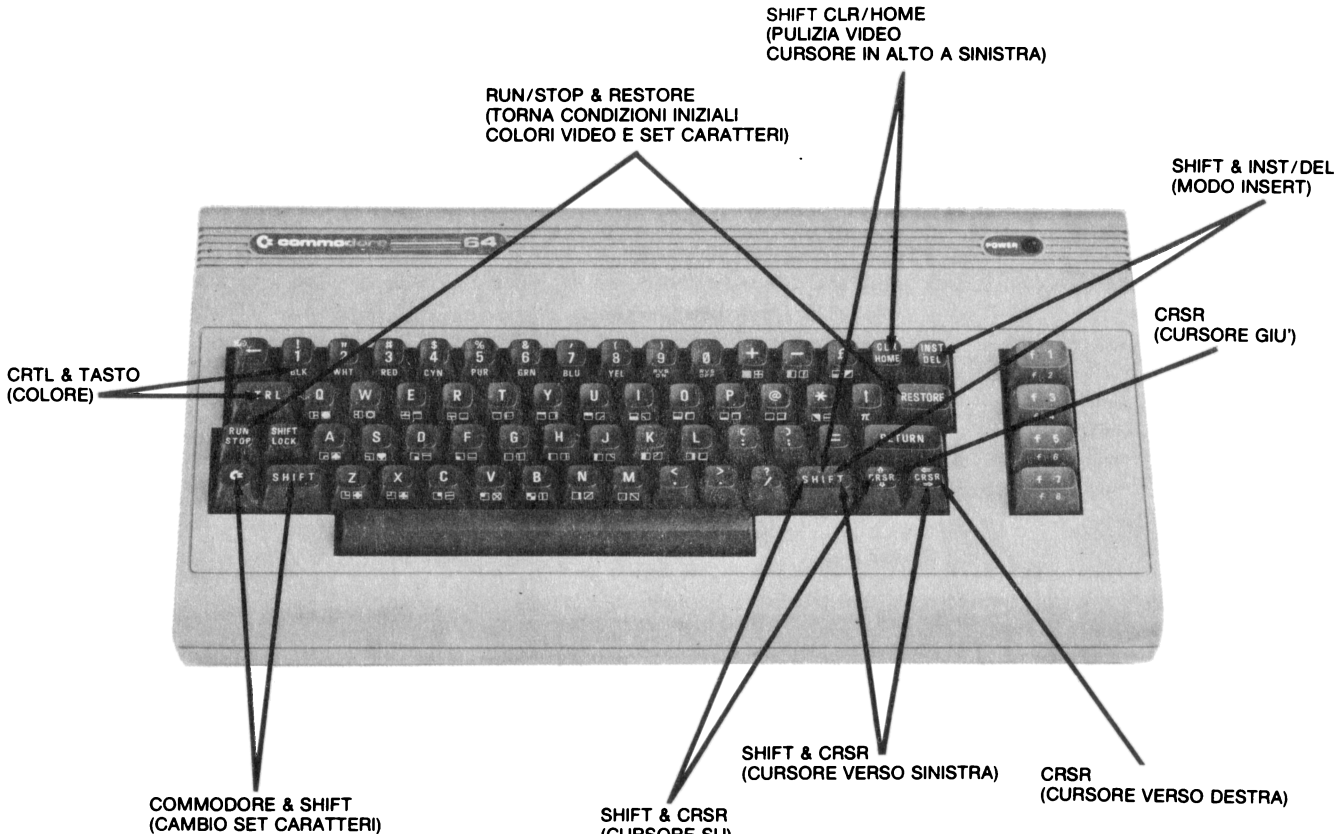
LA TASTIERA

3.1 LA TASTIERA

Il buffer della tastiera può contenere 10 caratteri esso si trova dal byte 631 al byte 640. Il contatore dei caratteri del buffer si trova nel byte 198, il codice dell'ultimo carattere premuto si trova nel byte 197. Porre dei caratteri ordinatamente nel buffer della tastiera, inserire in 198 il numero dei caratteri posti, corrisponde a dare la sequenza di caratteri manualmente sulla tastiera; puoi rivedere la routine RDEL al Paragrafo 4.6.
Puoi scrivere alcune linee di comandi in immediato sul video, partendo dalla posizione HOME, per esempio 5 linee, poi scrivi in immediato con POKE a partire dal byte 631 i codici: 19, 13, 13, 13, 13 ,13 e poi in 198 il numero 6. Le cinque linee vengono eseguite, infatti la tastiera (il suo buffer) viene scandita continuamente e vengono recepiti i caratteri che hai memorizzato; il primo manda a HOME e gli altri sono il RETURN per le linee che si trovano sul video.
La tastiera ha una immagine in memoria sotto forma di una matrice di 8 righe e 8 colonne, nella quale ogni posizione corrisponde a uno dei 64 tasti. Il registro 56320 contiene l'immagine delle linee della matrice tastiera e il registro 56321 quella delle colonne; analizzando il contenuto dei due registri il sistema stabilisce il tasto premuto e ne pone il codice in 197. Quando si preme insieme a un tasto uno dei tasti di controllo RVS, CTRL, COMMODORE o SHIFT il codice viene modificato.
Riteniamo utile presentare alcune immagini della tastiera evidenziando la funzione dei tasti.
Nel set maiuscolo/grafico i tasti normali danno le lettere maiuscole.


TASTI NORMALI
SET MAIUSCOLO GRAFICO
SHIFT & TASTO
TASTO NORMALE
RVS OFF
(CAMPO NORMALE)
RVS ON
(CAMPO INVERSO)
COMMODORE & TASTO
SHIFT & TASTO


SHIFT LOCK
(BLOCCO SHIFT)
HOME
(CURSORE
IN ALTO A SINISTRA)
RESTORE
INST/DEL
(NORMALE DEL)
CARATTERE SPAZIO
CONTROL
RUN/STOP
COMMODORE
TASTI
FUNZIONE
RETURN
MOVIMENTO
CURSORE ORIZZONTALE
SHIFT
MOVIMENTO
CURSORE VERTICALE


SHIFT CLR/HOME
(PULIZIA VIDEO
CURSORE IN ALTO A SINISTRA)
RUN/STOP & RESTORE
(TORNA CONDIZIONI INIZIALI
COLORI VIDEO E SET CARATTERI)
SHIFT & INST/DEL
(MODO INSERT)
CRSR
(CURSORE GIU')
CRTL & TASTO
(COLORE)
SHIFT & CRSR
(CURSORE VERSO SINISTRA)
CRSR
(CURSORE VERSO DESTRA)
COMMODORE & SHIFT
(CAMBIO SET CARATTERI)
SHIFT & CRSR
(CURSORE SU)

CAPITOLO 4

IL BASIC COMPILATO

4.1 IL BASIC COMPILATO

E' disponibile un programma di nome PETSPEED 64 - Basic Compiler, distribuito dalla COMMODORE, che consente di compilare programmi scritti in Basic. Il programma risiede su floppy disk ed è fornito di una ROM di protezione, senza la quale non funziona, da inserire nella Porta 2.

Compilare un programma significa avere come risultato un codice intermedio che non gira nel solito modo interpretativo, ma in un altro che è più veloce. Il programma compilato non lavora più in base al nome delle variabili o ai numeri delle linee Basic, ma in base agli indirizzi di memoria. Esso mantiene il nome del programma sorgente, con l'aggiunta di .WOW. Per caricare il programma compilato in memoria non è necessario avere inserita la chiave ROM di attivazione di PEETSPEED; basta scrivere:

```
LOAD"nome.WOW",8
```

e il programma viene caricato da disco. Per far girare il programma si scrive RUN, come al solito. Il nome del programma, dopo la compilazione, può essere modificato con la funzione RENAME del DOS.
Il programma compilato non è listabile; se si scrive LIST si vede una sola linea di programma. Inoltre esso non può essere interrotto per passare ad eseguire qualche comando in modo immediato; i nomi delle variabili non sono più direttamente accessibili. Il dischetto dei programmi di utilità che si genera nella procedura di compilazione, consente di ottenere utili prestazioni.
Il vantaggio del programma compilato sta nella sua maggior velocità esecutiva. Sul dischetto il programma compilato occupa molti più blocchi di quello originale, infatti esso comprende tutto, istruzioni e variabili e un suo particolare programma interprete.

Noi abbiamo usato il programma su un calcolatore collegato ad una sola unità 1541.

Nel manuale vengono spiegate alcune limitazioni che devono essere rispettate nei programmi da compilare; le riassumiamo brevemente:

.. le istruzioni DIM non accettano variabili per gli indici; si devono usare numeri;
.. i programmi non devono essere troppo lunghi; sembra che 70 blocchi o poco più siano già un limite;
.. il programma deve essere tutto in Basic e quindi non comprendere parti in linguaggio macchina;
.. non si possono fare riferimenti alle locazioni della pagina zero nel solito modo, ma il manuale fornisce le necessarie spiegazioni per modificare il programma;
.. non si può usare RUN numero-linea;
.. non si può usare LIST e SAVE nel programma;
.. un programma non può dividere le variabili con un altro programma da esso richiamato in memoria.

Il compilatore consente operazioni non accettate normalmente in Basic:

.. nei cicli FOR sono accettate variabili di controllo intere;
.. si possono definire funzioni utente che operano su stringhe.

Noi abbiamo preso il programma ORDIN del Capitolo 3 del secondo Volume; abbiamo apportato le poche modifiche necessarie, e cioè la linea 25 della DIM è stata cambiata dimensionando a 1000 numeri, invece che a N-1. Naturalmente così non si possono ordinare più di 1000 numeri e quindi avremmo dovuto aggiungere un controllo dopo la richiesta di N; non l'abbiamo fatto, dato che ci interessava solo un controllo dei tempi di esecuzione. Abbiamo fatto girare il programma originale e il programma compilato, dando lo stesso argomento negativo per l'origine dell'estrazione dei numeri a caso, e lo stesso numero di elementi da ordinare. I risultati sono i seguenti:

.. tra il compilato e il non compilato differiscono i numeri estratti, anche partendo dallo stesso argomento negativo; anche per il compilato la sequenza prodotta è sempre la stessa;
.. il compilato sembra più veloce; il tempo di estrazione è minore;
.. non possiamo fare il paragone dei tempi di ordinamento dato che i due programmi operano su sequenze diverse di numeri.

Riportiamo i risultati della prova dei due programmi, senza la stampa dei numeri.

RISULTATI ORDINA NON COMPILATO

```
BASE RND: -1984
ORDINAMENTO DI  57  NUMERI
TABELLA NUMERI ESTRATTI

TEMPO ESTRAZIONE 1.73333333  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 35.75  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 19.55  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 13.7666667  SEC.
```

RISULTATI ORDINA COMPILATO

```
BASE RND: -1984
ORDINAMENTO DI  57  NUMERI

TABELLA NUMERI ESTRATTI

TEMPO ESTRAZIONE .5  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 3.4  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 2.08333333  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 2.53333333  SEC.
```

Per curiosità riportiamo la tabella dei numeri estratti con argomento − 1984 e N = 57.

COMPILATO

```
TABELLA NUMERI ESTRATTI

 2538    7247    8859    9275    3603
 4016    7106    4361    1522    9440
 4309    6122    2491    7671    8966
 3684    3307    3512    1872    840
 2723    4425    4955    6565    9885
 5385    5418    1659    3636    2614
 4740    1285    125     8932    3063
 6456    7677    7093    7506    3274
 3852    2143    9194    2402    2580
 9945    1575    7864    2624    1813
 3701    1476    4510    354     7930
 7336    3036
TEMPO ESTRAZIONE .5  SEC.
```

NON COMPILATO

```
TABELLA NUMERI ESTRATTI

 1       265     747     883     664
 4072    2955    2889    9985    5283
 8191    4337    384     9694    3658
 5822    9693    2177    9935    4719
 1692    7346    346     8113    3688
 2220    7368    428     6759    7049
 3401    462     9638    3096    6887
 5263    1591    4968    778     6525
 5566    8869    5433    6397    7951
 1120    6169    4422    3728    4065
 7643    5136    2941    558     269
 3482    5984
TEMPO ESTRAZIONE 1.73333333  SEC.
```

Dopo aver esaminato i precedenti risultati, abbiamo deciso di provare a ordinare 300 numeri, ma gli stessi, con un programma Basic e con lo stesso programma compilato. Abbiamo modificato il programma ORDINA ed esso è diventato il programma ORDFISSO; in questo programma, invece di generare una serie di numeri a caso, viene letto un file sequenziale residente su disco, di nome DATI, che contiene 300 numeri. Il file DATI è stato preparato con un semplice programma e memorizzato su disco. In questo modo sia il programma compilato, che quello non compilato, lavorano sugli stessi numeri.
Segue la lista del programma ORDFISSO e i due risultati, solo per quanto riguarda i tempi di esecuzione.

```
1 REM ORDFISSO
2 REM ORDINAMENTO 300 NUMERI
5 REM LETTI DA UN FILE SEQUENZIALE
7 REM DI NOME: DATI
9 S$="          ":X$="ORDINATI CON METODO "
11 Y$="ORDINAMENTO"
14 N=300
17 OPEN4,4:CMD4
19 PRINT"ORDINAMENTO DI ";N;" NUMERI":PRINT
20 PRINT#4:CLOSE4
21 REM NUMERI MINORI UGUALI A 9999
23 INPUT"STAMPA SOLO TEMPI (S/N): ";R$
25 DIM N%(300),P%(300)
30 A=TI
31 OPEN10,8,10,"DATI,S,R"
40 FORK=0TON-1
45 INPUT#10,N%(K):P%(K)=N%(K)
50 NEXTK:B=TI:CLOSE10
55 A$="LETTI":B$="LETTURA DA DISCO":GOSUB100
60 C$=" 1":A$=X$+C$:B$=Y$
63 GOSUB200:GOSUB100
65 C$=" 2":A$=X$+C$:GOSUB500
67 GOSUB400:GOSUB100
69 C$=" 3":A$=X$+C$:GOSUB500
71 GOSUB300:GOSUB100
80 PRINT"BYTE LIBERI: ";FRE(0)
99 STOP
```

```
100 OPEN4,4:CMD4
105 PRINT"TABELLA NUMERI "A$:PRINT
107 IFR$="S"THEN125
110 IR=0:FORJR=0TON-1
115 N$=STR$(N%(JR)):N$=LEFT$(N$+S$,8)
117 PRINTN$;:IR=IR+1
120 IFIR=5THENPRINT:IR=0
123 NEXTJR:PRINT
125 PRINT"TEMPO "B$;(B-A)/60;" SEC."
130 PRINT#4:CLOSE4:RETURN
200 REM METODO A BOLLE
205 L=N-2:A=TI
210 J=0:FORK=0TOL
215 IFN%(K)<=N%(K+1)THEN225
220 C%=N%(K):N%(K)=N%(K+1):N%(K+1)=C%:J=1
225 NEXTK
230 IFJ=0THENB=TI:RETURN
235 L=L-1:GOTO210
300 REM METODO DIMEZZAMENTO INTERVALLO
305 IN=INT(N/2+0.5):A=TI
310 J=0:FORK=0TO(N-1-IN)
315 IFN%(K)<=N%(K+IN)THEN320
317 C%=N%(K):N%(K)=N%(K+IN):N%(K+IN)=C%:J=1
320 NEXTK
325 IFIN=1ANDJ=0THENB=TI:RETURN
327 IFIN=1THEN310
330 IN=INT(IN/2+0.5):GOTO310
400 REM METODO CICLI FISSI
405 K=0:A=TI
410 J=K+1:I=K
415 FORL=JTON-1
420 IFN%(K)<=N%(L)THEN430
425 K=L
430 NEXTL
435 IFK=ITHEN445
440 C%=N%(I):N%(I)=N%(K):N%(K)=C%
445 K=I+1:IFK=N-1THENB=TI:RETURN
450 GOTO410
500 FORK=0TON-1:N%(K)=P%(K):NEXTK:RETURN
```

RISULTATI ORDFISSO NON COMPILATO

```
ORDINAMENTO DI  300  NUMERI


TABELLA NUMERI LETTI

TEMPO LETTURA DA DISCO 9.5  SEC.


TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 979.233333  SEC.


TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 472.533334  SEC.


TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 472.433333  SEC.
```

RISULTATI ORDFISSO COMPILATO

```
ORDINAMENTO DI  300  NUMERI

TABELLA NUMERI LETTI

TEMPO LETTURA DA DISCO 4.7  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 92.95  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 53.3833334  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 50.2  SEC.
```

Dal confronto si vede che con la versione compilata:

.. il tempo di lettura da disco è dimezzato;
.. il tempo di ordinamento con il metodo 1 è circa 10 volte inferiore;
.. il tempo di ordinamento con il metodo 2 è circa 9 volte inferiore;
.. il tempo di ordinamento con il metodo 3 è circa 9 volte inferiore.

Non riportiamo il listato di ORDINA, dato che abbiamo indicato la modifica apportata rispetto a ORDIN.

4

INDICE

Capitolo 1 - LA PROGRAMMAZIONE IN ASSEMBLER
E IN LINGUAGGIO MACCHINA

1.1 Introduzione 1
1.2 Organizzazione della CPU 4
1.3 I modi di indirizzamento 11
1.4 Il set di istruzioni 14
1.4.1 Le operazioni logiche 16
1.4.2 Istruzioni di trasferimento 18
1.4.3 Operazioni logico matematiche 21
1.4.4 Istruzioni di controllo del flusso 25
1.4.5 Istruzioni sui flag 27
1.4.6 Operazioni sullo stack 28
1.5 Scrittura dei programmi in linguaggio macchina 29
1.5.1. Con PEEK e POKE 30
1.5.2 I monitor 33
1.5.3. Gli assemblatori 39
1.6 Interfaccia con il BASIC 39
1.7 La gestione delle interruzioni 43
1.8 Il sistema operativo del CBM 64 45
1.8.1 Il KERNAL 45
1.8.2 Il BASIC 57
1.9 Programmi esempio 63
1.9.1 RAMPEEK 64
1.9.2 Scrolling a destra 65
1.9.3 RESTORE num-linea 67
1.9.4 AT 70
1.9.5 BSAVE 71
1.9.6 Modifica INTERRUPT:FLASHING 73
1.9.7 Un paio di consigli 76
Tabella 1 - Istruzioni ASSEMBLER
in ordine alfabetico 79
Tabella 2 - Istruzioni ASSEMBLER
in ordine di codice operativo 82

Capitolo 2 - I FILE
2.1 Cosa è un file 85
2.2 Operazioni per la gestione dei file 91
2.3 Come si elaborano i file di dati 92

Capitolo 3 - FILE SU CASSETTA
3.1 Introduzione 95
3.2 File di dati 99
3.3 File di programmi 109
3.4 Esempio di file SEQUENZIALE 115

CAPITOLO 1

LA PROGRAMMAZIONE IN ASSEMBLER E IN LINGUAGGIO MACCHINA

1.1 INTRODUZIONE

Uno degli ostacoli più grossi nell'apprendimento della programmazione a BASSO LIVELLO è quello di far comunicare il calcolatore con l'esterno. L'ASSEMBLER è a basso livello perché vicino alla macchina, lo è ancora di più il LINGUAGGIO MACCHINA; alto livello è quello dei linguaggi vicini all'uomo, come BASIC, PASCAL, FORTRAN, ecc.. In questo capitolo vediamo come è fatta la CPU 6510 quali istruzioni è in grado di eseguire, come è fatto e come viene eseguito il programma.

Il programma che la CPU esegue è contenuto in memoria come una sequenza di byte. La CPU preleva dalla memoria il byte, che rappresenta l'istruzione da eseguire, e gli eventuali operandi indicati nei byte seguenti; esegue l'istruzione e passa a prelevare l'istruzione successiva. Ogni istruzione è quindi letta dalla memoria come un normale byte. Programmare in linguaggio macchina vuol dire porre nei byte di memoria i codici delle istruzioni che si vogliono eseguire, e gli operandi.

Come puoi immaginare, è assai laborioso scrivere un programma sotto forma di numeri (in binario), e per questo non è quasi mai usata questa tecnica.

Il linguaggio macchina lavora su dati numerici contenuti in un byte, quindi con valore da 0 a 255. Per poter trattare numeri più grandi o con segno si devono usare sottoprogrammi adatti. Lo stesso discorso vale per trattare numeri con cifre dopo il punto decimale (floating-point) e dati alfanumerici.

Le istruzioni che la CPU riconosce hanno ricevuto un nome mnemonico che si preferisce usare per scrivere il programma. Esistono dei programmi (alcuni chiamati ASSEMBLATORI), che traducono in codice oggetto (combinazione di zeri e uni da porre in memoria) il programma redatto da noi in forma mnemonica (che viene chiamato programma sorgente). Un'altra categoria di programmi, detti MONITOR, mettono a disposizione proprio un programma traduttore, anche se non si tratta di un vero assemblatore. Ne parliamo nel paragrafo 1.5.2.

Anche se, grazie ai MONITOR, non dobbiamo tradurre in numeri i codici delle istruzioni, abbiamo a che fare con i byte della memoria per gli operandi delle istruzioni e per gli indirizzi. Il sistema di numerazione decimale non consente una conversione immediata in binario. D'altra parte il sistema binario richiede troppe cifre per indicare un byte: ad esempio il numero 221 decimale corrisponde al numero binario 11011101. Il sistema di numerazione che è stato scelto per indicare gli operandi delle istruzioni è quello a base 16, o ESADECIMALE. In tale sistema vi sono 16 cifre: 0 1 2 3 4 5 6 7 8 9 A B C D E F. Il vantaggio dell'esadecimale rispetto al decimale è che un numero esadecimale si può tradurre in binario più in fretta, perché ogni cifra rappresenta 4 bit; rispetto al binario, il vantaggio è che i numeri sono più compatti.

In questo libro e nei libri di ASSEMBLER 6502 i numeri esadecimali sono preceduti dal segno dollaro: il numero $23 si rappresenta in binario 00100011.

Nella Figura 1.1 trovi la corrispondenza dei valori delle cifre esadecimali nei sistemi binario e decimale. Come nel sistema decimale dopo il numero 9 si passa a 10 ponendo 1 nella posizione più a sinistra, così nell'esadecimale dopo $F si passa a $10, che vuol dire 16 in decimale. La Figura 1.2 mostra ancora un esempio di

ESADECIMALE	BINARIO	DECIMALE
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Figura 1.1 Esadecimale, binario e decimale

BASE 2			
1	0	1	
2^2	2^1	2^0	
4	2	1	
4 +	0 +	1	=
		5	

BASE 16			
1	0	1	
16^2	16^1	16^0	
256	16	1	
256 +	0 +	1	=
		257	

BASE 10			
1	0	1	
10^2	10^1	10^0	
100	10	1	
100 +	0 +	1	=
		101	

Figura 1.2 Analisi del numero 101

conversione: lo stesso numero 101 viene analizzato prima come binario, poi come esadecimale e poi come decimale. A titolo informativo aggiungiamo che nel mondo dei calcolatori si fa spesso uso anche del sistema a base 8, detto OTTALE, dove le cifre vanno da 0 a 7. Noi non faremo mai riferimento a tale sistema in questo volume, e per questo non lo approfondiamo.

Accenniamo, da ultimo, al sistema BCD (Bynary Coded Decimal), cioè decimale a codifica binaria: è una via di mezzo tra esadecimale e decimale, perché ogni cifra BCD rappresenta 4 bit (come l'esadecimale) e le cifre vanno da 0 a 9 (come il decimale). Il difetto di questo sistema di rappresentazione è un notevole spreco di bit: un byte può infatti contenere solo 100 combinazioni ammesse (da 0 a 99), invece di 255 possibili in binario.
Un altro svantaggio risiede nell'elaborazione. Consideriamo come esempio una qualsiasi operazione di somma:

NUMERO	RAPPRESENTAZIONE BCD	
08 +	00001000	+
12 =	00010010	=
20	00100000	invece di 00011010 come deve essere.

Osserviamo subito che l'unità di calcolo all'interno della CPU deve lavorare diversamente se gli operandi e il risultato sono in BCD da come opera con gli operandi in binario. Nella 6510 è possibile selezionare il modo BCD con un'istruzione, e tornare al modo binario con un'altra istruzione, e questa possibilità aiuta notevolmente il programmatore ASSEMBLER nel gestire i dati decimali.

1.2 ORGANIZZAZIONE DELLA CPU 6510

La 6510 è una CPU che appartiene alla famiglia 6500, una serie di unità centrali di elaborazione sviluppata negli anni '70 negli USA dalla ROCKWELL INTERNATIONAL CORPORATION. La "filosofia" dei progettisti è stata quella di creare una CPU dotata di un numero di istruzioni relativamente limitato, un numero di registri interni anch'esso limitato, ma di potenti e veloci modi di indirizzamento.

Per limitare il numero di istruzioni non si sono poste istruzioni specifiche per gestire l'I/O: la CPU gestisce solo la memoria, e l'I/O deve essere collegato alla CPU come la memoria.
Il limitato numero di registri è compensato dalla gestione rapida dei dati in PAGINA ZERO: le celle di memoria che partono dall'indirizzo 2 fino a $00FF sono da considerarsi come un'estensione dei registri interni della CPU, poiché si possono indirizzare più rapidamente e permettono indirizzamenti assai potenti, come vedremo nel prossimo paragrafo.

Osserviamo la Figura 1.3 per vedere come funziona internamente la CPU. Partiamo dai registri: la 6510 contiene 6 registri a disposizione del programmatore.
Ogni registro è formato da 8 bit (tranne il PROGRAM COUNTER che ha 16 bit), ed ha la caratteristica di poter immagazzinare o fornire alla propria uscita un dato di 8 bit. I registri non elaborano le informazioni, ma semplicemente "ricordano" lo stato degli 8 bit di cui sono composti. I registri sono i seguenti:

- Index register Y: REGISTRO INDICE Y
- Index register X: REGISTRO INDICE X
- Stack pointer register: STACK POINTER
- Accumulatore: ACCUMULATORE
- PCL: metà del PROGRAM COUNTER, gli 8 bit meno significativi
- PCH: metà del PROGRAM COUNTER, gli 8 bit più significativi
- Processor status register: registro che contiene 7 bit; ognuno di questi bit, chiamati in gergo FLAG, ha un significato che riflette lo stato interno della CPU; esso dipende dalle istruzioni eseguite o dal risultato dell'ultima operazione. Dei FLAG parleremo più dettagliamente nel seguito di questo paragrafo.

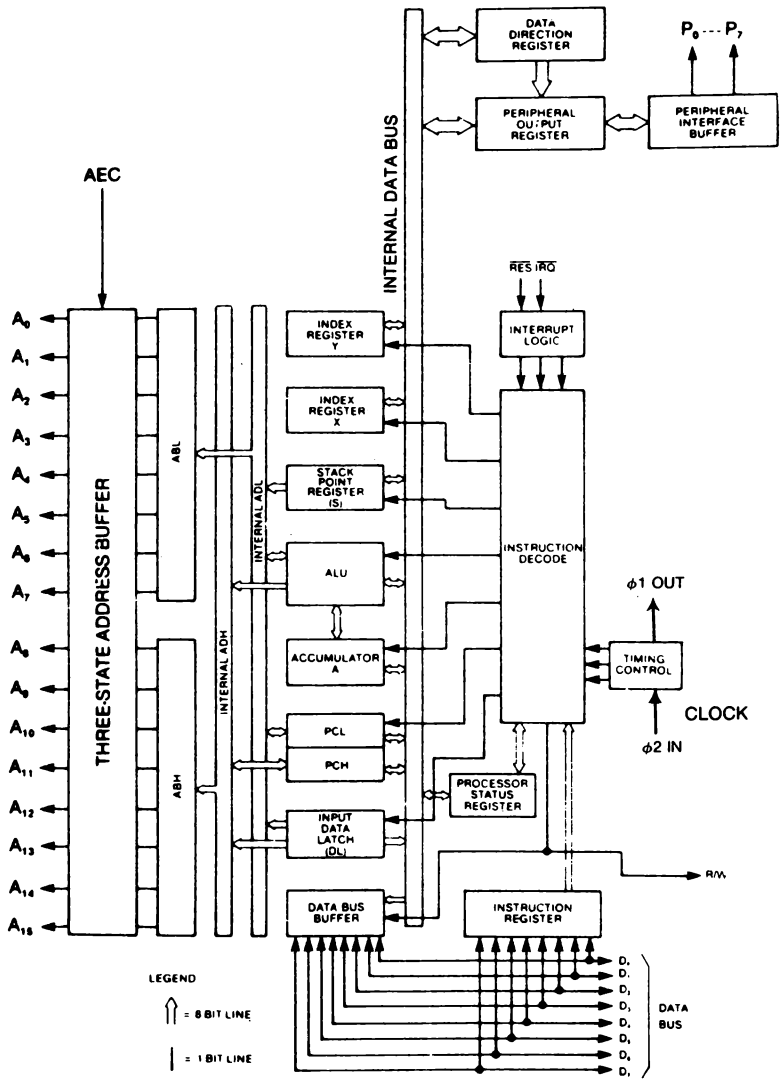


Figura 1.3 Architettura interna della CPU 6510

Tutti i registri sono collegati ad un BUS DATI interno, e attraverso questo BUS (insieme di 8 fili che corrispondono ciascuno a un bit) possono caricare o scaricare dati. Ogni registro risulta collegato anche a una parte della CPU indicata nello schema come INSTRUCTION DECODE (decodifica dell'istruzione): questo blocco contiene l'unità di controllo della CPU, unità che controlla tutte le abilitazioni dei registri (può comandare a un registro di caricare il dato presente sul BUS DATI interno, o di presentare su tale BUS il proprio contenuto) e l'ALU (unità aritmetico-logica). La decodifica dell'istruzione è la parte più complessa di tutta la CPU: la sua funzione è quella di generare una sequenza di abilitazione dei registri in modo da eseguire l'istruzione specificata dal programma (il codice dell'istruzione da eseguire viene posto nell'INSTRUCTION REGISTER, che lo presenta al blocco INSTRUCTION DECODE, come puoi osservare nella parte in basso a destra dello schema).

Abbiamo nominato l'ALU; puoi osservare nello schema che tale unità è collegata al BUS DATI interno, all'ACCUMULATORE, e ai due BUS interni ADL e ADH, che sono la parte bassa (ADL, 8 bit meno significativi) e parte alta (ADH, 8 bit più significativi) del BUS INDIRIZZI interno. L'ALU è in grado di eseguire operazioni aritmetico-logiche tra i vari BUS a cui è collegata. Ad esempio l'ALU è in grado di eseguire la somma tra il contenuto del BUS DATI interno e l'accumulatore, e memorizzare il risultato. Essa è in grado poi di presentare il risultato memorizzato sul bus dati; tutte le operazioni che l'ALU esegue sono comandate dalla sezione INSTRUCTION DECODE.

Vediamo ora come la CPU esegue un'istruzione, ad esempio l'istruzione:

LDA #$24

che si trova nei byte di memoria $1000 e $1001:

$1000 = $A9
$1001 = $24

L'istruzione LDA # produce l'effetto di caricare nell'accumulatore il dato che segue. Il codice di questa istruzione è 10101001 binario, cioè $A9.
Il PROGRAM COUNTER (che da ora chiameremo P.C.) contiene l'indirizzo dell'istruzione che deve essere eseguita, cioè nel nostro caso $1000. Il contenuto del P.C. viene presentato sul BUS INDIRIZZI, si attende una transizione di $\phi 1$ in modo che la memoria abbia il tempo di fornire il dato richiesto, e si carica il contenuto del BUS DATI esterno (in basso nel disegno) nel registro delle istruzioni. Nel nostro caso tale registro contiene il contenuto del byte $1000, cioè $A9. INSTRUCTION DECODE (da ora I.D.) riconosce che si tratta dell'istruzione LDA #, ed esegue tutte le abilitazioni necessarie, in particolare:

- PCL (che contiene $00) scarica su DATA BUS interno.
- ALU incrementa DATA BUS ($00) e presenta il risultato ($01) su ADL interno.
- PCL carica da ADL ($01): è stato incrementato il PROGRAM COUNTER.
- Il BUS indirizzi contiene ora $1001. I.D. attende la transizione di $\phi 1$ per accedere all'indirizzo di memoria specificato.
- Passata la transizione, I.D. disabilita l'uscita dell'ALU su ADL e l'uscita di PCL sul BUS DATI interno, abilita DATA BUS BUFFER a presentare sul BUS DATI interno il contenuto del BUS DATI esterno ($24, il contenuto del byte $1001).
- I.D abilita l'accumulatore a caricare dal BUS DATI interno ($24). È stato caricato $24 nell'accumulatore.
- I.D. disabilita l'uscita dal DATA BUS BUFFER, abilita PCL ($01) a scaricare sul BUS DATI interno e l'ALU incrementa il contenuto del BUS DATI interno, presentando il risultato ($02) su ADL.
- PCL viene abilitato a caricare da ADL, e intanto viene effettuato un accesso alla locazione $1002 (si attende una transizione di $\phi 1$) per leggere il codice dell'istruzione successiva.

Non ti spaventare se hai trovato difficile seguire tutte queste operazioni: abbiamo affrontato un argomento abbastanza complicato. Non è essenziale capire perfettamente come funziona dentro la CPU per usarla, tuttavia pensiamo sia utile avere almeno un'idea di come si svolgono le operazioni all'interno di questa "scatola nera". Puoi soffermarti su questi argomenti in una seconda lettura: ti consigliamo di seguire sulla Figura 1.3 tutte le operazioni, se vuoi capire l'esempio appena spiegato: in seguito puoi anche provare per esercizio (e per divertimento) a far eseguire alla CPU qualche istruzione.

Soffermiamo ora l'attenzione sui FLAG. Abbiamo accennato che vi è un registro a 8 bit chiamato STATO DEL PROCESSORE che contiene 7 FLAG. Un FLAG è un bit che ha un certo significato logico. FLAG in inglese vuol dire "bandiera": è come una bandiera che si alza ad indicare che un determinato evento si è verificato.

Nella Figura 1.4 trovi riassunti i nomi dei registri della 6510. Puoi osservare come sia organizzato il REGISTRO DI STATO:

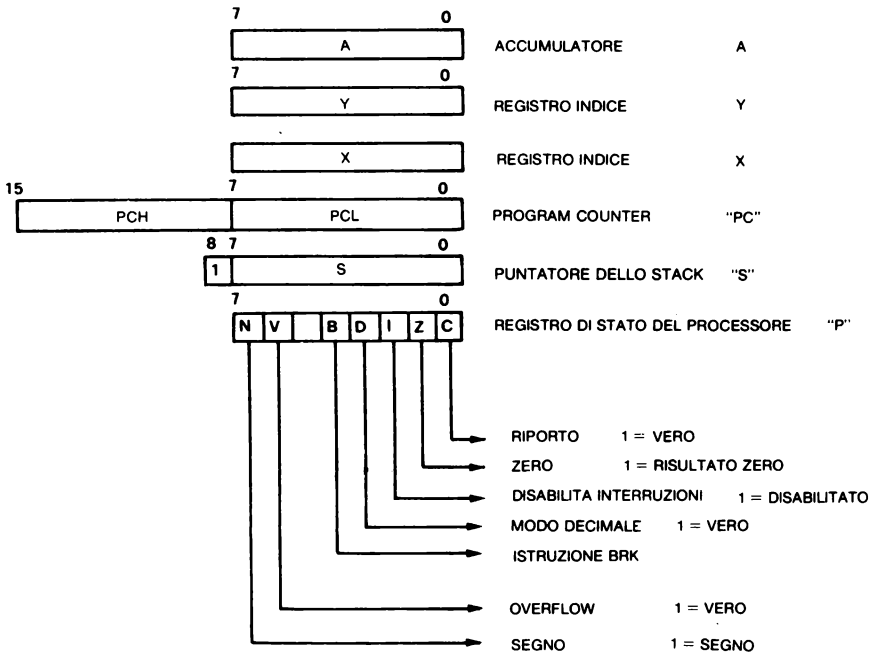


Figura 1.4 Registri della CPU 6510 a disposizione del programmatore

● BIT 7 = N. Flag di SEGNO. Questo bit è a 1 se il risultato dell'operazione è negativo. La 6510 tratta i numeri negativi in COMPLEMENTO A DUE. Secondo tale rappresentazione il bit più significativo rappresenta il segno, (0=positivo, 1=negativo). I numeri negativi sono rappresentati come "parte che manca al valore assoluto del numero per arrivare a 256" (256 nei numeri a 8 bit; 65536 nei numeri a 16 bit; 2^n nei numeri a n bit). Prendiamo il numero —12. Quanto manca da 12 per arrivare a 256? Manca 244; 244 si rappresenta in binario puro 11110100. Questa è anche la rappresentazione di —12 in complemento a 2. Tale rappresentazione a prima vista appare assai scomoda, ma è in realtà la più efficiente dal punto di vista dell'elaborazione. Il FLAG DI SEGNO riporta lo stato del bit più significativo del risultato. Il suo valore viene aggiornato dopo le istruzioni di somma, sottrazione, incremento, decremento, caricamento da memoria, confronto, rotazione di bit, scorrimento di bit, trasferimento da registro a registro, ecc. La tabella 1 a fine Capitolo mostra nella sesta colonna da destra quali sono le istruzioni che alterano lo stato di questo FLAG.

● BIT 6 = V. Overflow. Questo bit indica se c'è stato traboccamento nelle operazioni in complemento a 2, cioè se il risultato è troppo grande (> 27) o troppo piccolo (<— 128) per essere rappresentato in complemento a 2 con 8 bit. In questo caso il risultato è sbagliato. Nell'aritmetica in valore assoluto e nei byte meno significativi dei numeri in complemento a 2 che occupano più di un byte, lo stato di questo bit non ha nessun significato. Solamente somma e sottrazione alterano lo stato di questo FLAG, oltre all'istruzione BIT (in questo caso V assume un significato diverso, e indica lo stato del bit 6 del risultato), e CLV che pone a 0 lo stato di questo bit.

● BIT 5 = non utilizzato

● 4 = B. Istruzione BRK. L'istruzione BRK è molto particolare: quando viene incontrata la CPU esegue le stesse operazioni di quando riceve un interrupt (vedi Paragrafo 1.6), con la differenza che il registro di stato salvato nello stack contiene 1 nella posizione del bit di BREAK. Normalmente nella programmazione non viene usato. Poiché non esistono istruzioni che permettono di vedere lo stato di questo FLAG nella CPU, bisogna prelevarlo dallo stack con una piccola routine. Puoi vedere questa routine disassemblando il sistema operativo, partendo dall'indirizzo $FF48. Vengono salvati nello STACK i registri Δ, x e y poi, se il FLAG di BRK è a 1, il SISTEMA OPERATIVO fa un salto indiretto a $0316, se no a $0314.

● BIT 3 = D. Modo decimale. Abbiamo visto nel precedente paragrafo che rappresentando i numeri col modo BCD (decimale a codifica binaria) le operazioni aritmetiche devono essere eseguite con un criterio diverso rispetto al binario puro. Questo FLAG, che è controllato dal programmatore, informa l'ALU che i conti vanno fatti in BCD. L'istruzione SED attiva il modo BCD, e CLD lo disattiva.

● BIT 2 = I. Disabilita le interruzioni: questo bit è sotto il controllo diretto del programmatore, cioè non cambia in base al risultato di operazioni; lo si può porre a 1 o a 0, l'istruzione SEI lo pone a 1, l'istruzione CLI lo pone a 0. Come tutti gli altri FLAG il suo valore viene aggiornato anche dopo le istruzioni PLP e RTI, che prelevano lo stato del processore dallo STACK.

● BIT 1 = Z. Risultato zero. Questo è uno dei FLAG più usati nei programmi in ASSEMBLER: indica se tutti gli 8 bit del risultato sono a 0, e solo in tal caso Z vale 1. Come per il FLAG N, anche il FLAG Z viene modificato da numerose istruzioni, come puoi osservare nella quinta colonna da destra della tabella 1.

● BIT 0 = C. Carry, o riporto, o prestito. Questo FLAG viene alterato dal risultato di diverse operazioni, e talvolta con significati diversi. Nella somma, indica se il risultato ha superato 255, e nella sottrazione indica se c'è stato prestito. Puoi vedere quali operazioni interessano il FLAG C nella quarta colonna da destra della tabella 1.

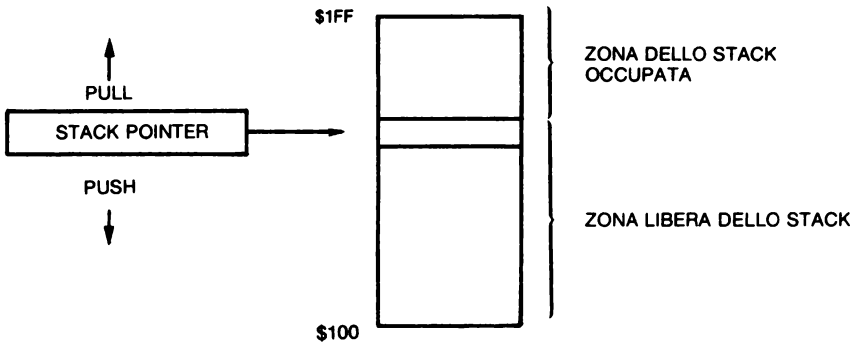


Figura 1.5 STACK e STACK POINTER

Abbiamo nominato i termini STACK e STACK POINTER, senza descrivere il loro significato: lo STACK è un'area di memoria riservata alla CPU, dove la CPU pone delle informazioni in maniera temporanea. STACK vuol dire "catasta", e il suo funzionamento è paragonabile a quello di una catasta o, più elegantemente, di una struttura di dati di tipo LIFO (Last In, First Out, cioè l'ultimo dato entrato è il primo che esce). Nella 6510 lo STACK occupa le locazioni da $100 a $1FF, cioè la pagina 1 della memoria. Lo STACK POINTER è un registro della CPU di 8 bit, che indica la parte meno significativa del primo byte libero dello STACK (la parte più significativa è sempre $01). All'accensione lo STACK POINTER viene posto a —$FF. Ogni volta che si spinge un dato nello STACK (PUSH) il dato viene posto all'indirizzo specificato dallo STACK POINTER, dopodiché lo STACK POINTER viene decrementato.

Nel prelevare un dato dallo STACK (PULL) viene prima incrementato lo STACK POINTER e poi prelevato il dato.

Lo STACK è molto importante per il funzionamento del calcolatore: l'istruzione JSR (salta alla subroutine), ad esempio, usa lo STACK per memorizzare l'indirizzo dove tornare al momento dell'istruzione RTS, ed anche l'INTERRUPT fa uso dello STACK per poter riprendere il programma alla fine della routine di servizio

dell'interrupt. Vi sono diverse istruzioni che fanno uso dello STACK, come PHP, PLP, PHA, PLA, RTS, RTI, BRK; inoltre lo STACK POINTER può essere trasferito nel registro X (TSX) e viceversa (TXS). Vedi il Paragrafo 1.4 per la descrizione del funzionamento di queste istruzioni.

Abbiamo già visto la funzione dei registri STATO del processore, STACK POINTER, PROGRAM COUNTER. Vediamo ora l'ACCUMULATORE.
Come puoi notare dalla Figura 1.3, l'ACCUMULATORE è collegato direttamente all'ALU. È questo il registro a cui fa riferimento gran parte delle istruzioni aritmetico-logiche. Sull'accumulatore è possibile effettuare scorrimento di bit, operazioni logiche AND, OR, EOR, (Exclusive OR), somma, sottrazione.
I due registri indice X e Y sono usati come indici negli indirizzamenti indicizzati, come puoi meglio osservare nel prossimo paragrafo.

1.3 I MODI DI INDIRIZZAMENTO

Nella descrizione della 6510 abbiamo detto che tale CPU ha molti modi di indirizzamento, e molto potenti; ora passiamo a descriverli.
Il modo di indirizzamento in una CPU è il modo con cui è possibile ricavare l'indirizzo dell'operando.
Poiché ogni istruzione ha lo scopo di produrre quache effetto su qualche operando (registro della CPU o cella di memoria), possiamo dedurre che ogni istruzione (anche NOP, che fa cambiare il PROGRAM COUNTER per eseguire l'istruzione seguente) usa un modo di indirizzamento. Nella prima riga della tabella 1 sono riportati tutti i modi di indirizzamento della 6510. VEDIAMOLI UNO PER UNO.

● IMMEDIATO. L'operando segue l'istruzione. Si indica con il segno # prima dell'operando. Ad esempio:
LDA #=24
si codifica in memoria così:
$A9 $24
Nell'accumulatore viene posto il numero $24.

● ASSOLUTO. L'operando è contenuto in memoria all'indirizzo specificato dai due byte seguenti il codice dell'istruzione:
LDA $134F
si codifica in memoria così:
$AD $4F $13
Nell'accumulatore viene posto il numero contenuto nel byte di indirizzo $134F (prima il byte basso e poi quello alto).

● PAGINA ZERO. L'operando è contenuto in memoria all'indirizzo della pagina 0 specificato dal byte che segue l'istruzione:

LDA $24

si codifica in memoria così:

$A5 $24

A differenza dell'immediato, nell'accumulatore viene posto il byte il cui indirizzo è contenuto nel byte di indirizzo $0024 (e non il numero $24).

● ACCUMULATORE. L'operando è dentro la CPU, e precisamente nell'accumulatore:

LSR

si codifica in memoria così:

$4A

Viene fatto scorrere verso destra il contenuto dell'accumulatore.

● IMPLICITO. L'operando è sottinteso dal tipo di istruzione:

CLC

si codifica in memoria così:

$18

Azzera il CARRY. L'operando è palesemente il CARRY.

● ASSOLUTO, X. L'operando è contenuto in memoria, e l'indirizzo si ricava sommando ai due byte, che seguono il codice dell'istruzione, il contenuto del registro X:

LDA $103D,X

si codifica in memoria così:

$BD $3D $10

Supponiamo che il registro X contenga $D1: $103D è sommato a $D1, ottenendo $110E. L'operando è contenuto all'indirizzo $110E.

● ASSOLUTO, Y. Opera come ASSOLUTO, X, ma anziché il registro X usa il registro Y:

LDA $103D,Y

si codifica in memoria così:

$B9 $3D $10.

● PAGINA 0, X. L'operando si trova in pagina 0, all'indirizzo ottenuto sommando il byte seguente l'istruzione e il registro X:

LDA $F0,X

si codifica in memoria così:

$B5 $F0

Se X contiene $32, nell'accumulatore viene posto il byte contenuto nella cella $0022 ($F0 + $32 = $122, ma l'operando viene prelevato sempre dalla pagina 0).

● PAGINA 0, Y. Opera come PAGINA 0, X, ma usa il registro Y. È implementato solo con le istruzioni LDX e STX:

LDX $20,Y

si codifica in memoria così:

$B6 $20.

● INDIRETTO. L'istruzione è seguita da due byte che indicano l'indirizzo del puntatore dove si trova l'indirizzo dell'operando:

JMP ($2000)

si codifica in memoria così:

$6C $00 $20

Produce un salto non alla cella $2000, ma all'indirizzo specificato nelle celle $2000 e $2001. Se tali celle contengono i numeri $12 e $34 rispettivamente, la CPU salta all'indirizzo $3412 (il byte più significativo è sempre dopo quello meno significativo). È usato solo dall'istruzione JMP.

● (INDIRETTO), Y. Si chiama INDIRETTO INDICIZZATO, ed è uno dei modi più potenti. L'istruzione è seguita da un byte, che indica l'indirizzo in pagina 0 del puntatore al dato. All'indirizzo contenuto nel puntatore va sommato il contenuto del registro Y per ricavare l'indirizzo dell'operando:

LDA ($03),Y

si codifica in memoria così:

$B1 $03

Se le celle $0003 e $0004 contengono rispettivamente $14 e $2E e il registro Y contiene $F1, l'indirizzo che risulta è il seguente: $2E14 + $F1 = $2F05. Viene caricato l'accumulatore col contenuto della cella $2F05.

● (INDIRETTO), X. Si chiama INDICIZZATO INDIRETTO, e differisce dal precedente. Il contenuto del registro X è sommato al byte seguente l'istruzione, il risultato indica l'indirizzo di un puntatore in pagina 0. L'operando si trova all'indirizzo specificato dal puntatore suddetto:

LDA ($F0),X)

si codifica in memoria così:

$A1 $F0

Se X contiene $31 viene sommato $F0 con $31, ottenendo $121. È considerata la parte meno significativa, e l'indirizzo dell'operando è prelevato dalle celle $21 e $22.

● RELATIVO. Questo indirizzamento è usato per tutte le istruzioni di salto condizionato, o BRANCH. L'operando, che segue l'istruzione, è un byte considerato come un numero in complemento a due (può variare da —128 a +127). L'operando è sommato al PROGRAM COUNTER e il risultato ottenuto è immagazzinato nel PROGRAM COUNTER. L'operando indica lo "spiazzamento" da fornire al PROGRAM COUNTER nel caso in cui si verifica la condizione di salto.

BCC $09

si codifica in memoria così:

$90 $09

e produce l'effetto seguente: se il CARRY é a 1 prepara il P.C. per eseguire l'istruzione che segue, se invece CARRY = 0 al P.C viene sommato 9 rispetto sempre all'istruzione che segue, e il programma riprende dal nono byte che segue l'istruzione dopo BRANCH (ma non necessariamente dalla nona istruzione, perché una istruzione può occupare uno, due o tre byte). Nota che usando un MONITOR devi fornire l'indirizzo assoluto, e il MONITOR calcola l'indirizzo relativo. Il vantaggio di questo tipo di indirizzamento è che puoi trasferire il programma in un'altra zona della memoria senza cambiare gli indirizzi dei salti.

Abbiamo analizzato tutti i modi possibili di indirizzamento della CPU 6510. Non occorre impararli tutti a memoria subito, ma ti consigliamo di rileggere quelli che non hai capito prima di avventurarti nella scrittura di programmi in linguaggio ASSEMBLER.

1.4 IL SET DI ISTRUZIONI

Nell'esposizione del completo set di istruzioni del microprocessore 6510, distingueremo cinque categorie principali

1) trasferimento dati,
2) logico matematiche,
3) controllo del flusso,
4) manipolazione dei flag,
5) uso dello stack.

● 1): Consentono di trasferire dati di 8 bit da un registro all'altro, da un registro alla memoria o viceversa.

● 2): Permettono al microprocessore di eseguire operazioni aritmetiche (più e meno), operazioni logiche (AND, OR, EOR (exclusive OR)), operazioni di scorrimento (shift e rotazioni), incremento e decremento.

● 3): Sono usate per i salti condizionati, salti non condizionati, salti a subroutine e gestione dell'interrupt.

● 4): Consentono di agire sul registro di stato del processore.

● 5): Permettono di introdurre ed estrarre dati dallo stack, senza doverti preoccupare della gestione dello stack pointer.

Nel corso dell'esposizione compaiono alcuni simboli di cui riportiamo il significato:

A: accumulatore
M: cella di memoria (byte di memoria)
P: registro di stato
S: stack pointer
X: registro X
Y: registro Y
DATO: dato specificato
PC: program counter
STACK: l'ultima cella dello stack
=: assegnamento
$\wedge$: AND logico
V: OR loico
$\forall$: EOR (OR esclusivo)
$\overline{C}$: NOT CARRY
(...): contenuto di ...
BN: bit N
MN: bit N della cella (byte) di memoria specificata.
V: FLAG di overflow
N: FLAG di segno
I: FLAG di interrupt
D: FLAG decimale
C: FLAG di carry
Z: FLAG di zero

1.4.1 Le operazioni logiche

Prima di illustrare le istruzioni del set della CPU 6510, vorremmo spendere qualche parola riguardo alle operazioni logiche AND OR ed EOR (exclusive OR).
Hai già incontrato le prime due in BASIC e le hai usate per fare di più condizioni una sola condizione, ad esempio:

IF (A AND Z<1) OR A$="PIPPO" THEN ...

l'istruzione dopo il THEN viene eseguita se sono vere entrambe le prime due condizioni o se è vera la terza: cioè se è vera la condizione composta.
Ma le operazioni logiche AND e OR sono più generali di quanto non possa sembrare dall'esempio appena fatto: esse possono agire, infatti, anche tra numeri interi, oltre che tra condizioni vere o false.

Cominciamo a definire le operazioni logiche tra bit:

0 AND 0 = 0	0 OR 0 = 0	0 EOR 0 = 0
0 AND 1 = 0	0 OR 1 = 1	0 EOR 1 = 1
1 AND 0 = 0	1 OR 0 = 1	1 EOR 0 = 1
1 AND 1 = 1	1 OR 1 = 1	1 EOR 1 = 0

Il risultato della AND tra due bit è quindi uguale a 1 solo se il primo E il secondo bit sono uguali a 1; il risultato della OR tra due bit è uguale a uno se lo sono il primo O il secondo (o entrambi); il risultato della EOR tra due bit è uguale a 1 se lo sono il primo O il secondo (ma non entrambi).

Estendere ora il significato dell'operazione a numeri interi di X bit è semplice: il bit N del risultato di un'operazione logica tra byte è uguale al risultato dell'operazione logica tra i bit corrispondenti dell'operando; ad esempio:

2 AND 3 = 2
infatti:
00000010 AND
00000011 =
00000010 cioè 2

Le AND e OR del BASIC sono proprio le operazioni logiche che abbiamo descritto.

Prova a chiedere al tuo COMMODORE 64:

PRINT 2 AND 3

e vedrai che il risultato sarà corretto. Queste operazioni vanno bene anche per legare tra di loro condizioni perché il COMMODORE 64 assegna ad un'espressione vera il valore —1 (in complemento a due su 16 bit = 16 bit a 1) e ad una falsa il valore 0 (16 bit a 0) (prova a scrivere PRINT 2>7 o PRINT 2<7): quando deve combinare tra loro due condizioni con la funzione AND il risultato è —1 (cioè vero) solo se entrambe le condizioni valgono —1 (cioè sono vere); se l'operazione è la OR il risultato è —1 se almeno una delle due vale —1.

A questo punto sorge spontanea una domanda: perché nel set di istruzioni di un microprocessore appaiono queste strane operazioni mentre non compaiono operazioni più usuali come la moltiplicazione e la divisione? Perché queste operazioni ti permettono di leggere o alterare un singolo bit o un gruppo di bit di un byte, cosa questa assai più importante, in molti casi, di una moltiplicazione o una divisione (che si possono comunque ottenere grazie a un opportuno algoritmo). Ad esempio: vuoi sapere quanto vale il bit 3 di un certo byte? Basta fare la AND tra questo byte e 8 (2^3): se il risultato è 0 il bit vale 0, se è 8 il bit è a 1. Altro esempio: vuoi porre a 1 il bit 5? Fai la OR con 32 (2^5). Se lo volevi a 0? Fai la AND con 223 (255-32 cioè un numero che ha tutti i bit a uno tranne il quinto). La EOR a cosa serve? Serve a cambiare il valore di uno o più bit. Ad esempio 01010011 EOR 00001111 = 01011100: questa operazione ha cambiato il valore dei bit 3-0 del primo operando (cioè quelli che valgono 1 nel secondo operando). Se fai una EOR tra il contenuto di un byte e $FF "neghi" il contenuto del byte: cioè i bit che valevano 1 valgono 0 e viceversa.

Ti proponiamo il programma AND&OR che genera operazioni logiche casuali tra numeri binari di 8 bit. Dopo aver calcolato la risposta premi un tasto e il tuo calcolatore ti darà la conferma.

```
0   REM AND&OR
10  A=INT(RND(0)*256):B=INT(RND(0)*256)
15  OP=INT(RND(0)*3)
20  PRINTCHR$(147);
30  PRINT:PRINT:PRINT"     ";: FORI=0TO7
40  PRINTMID$("o●",1-((A AND 2↑(7-I))>0),1);
50  NEXTI:PRINT TAB(12)A
60  PRINTMID$("AND OREOR",OP*3+1,3);
70  PRINT" ";: FORI=0TO7
80  PRINTMID$("o●",1-((B AND 2↑(7-I))>0),1);
90  NEXTI:PRINT TAB(12)B
95  PRINT "     --------"
100 PRINT"  = ";
110 IF OP=0 THEN R=A AND B : GOTO 130
115 IF OP=1 THEN R=A OR B  : GOTO 130
120 R=(A OR B) AND (NOT A OR NOT B)
130 GETA$:IFA$="" THEN 130
140 FORI=0TO7
150 PRINTMID$("o●",1-((R AND 2↑(7-I))>0),1);
160 NEXTI:PRINT TAB(12)R
170 GETA$:IFA$="" THEN 170
180 IF A$<>"←" THEN RUN
```

1.4.2 Istruzioni di trasferimento

LDA: LoaD Accumulator A=DATO

Il dato specificato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — indicizzato X e Y — pagina 0 indicizzato X — indiretto indicizzato — indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

LDX: LoaD X register X=DATO

Il dato specificato viene posto nel registro X.
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — indicizzato Y — pagina 0 indicizzato Y.
FLAG MODIFICATI: zero e segno.

LDY: LoaD Y register Y=DATO

Il dato specificato viene posto nel registro Y.
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — indicizzato X — pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

STA: STore Accumulator M=(A)

Il contenuto dell'accumulatore viene posto nella cella di memoria di indirizzo specificato. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: assoluto — pagina 0 — indicizzato X e Y — pagina 0 indicizzato X — indiretto indicizzato — indicizzato indiretto.
FLAG MODIFICATI: nessuno.

STX: STore X register M=(X)

Il contenuto del registro X viene posto nella cella di memoria di indirizzo specificato. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: assoluto — pagina 0 — pagina 0 indicizzato Y.
FLAG MODIFICATI: nessuno.

STY: STore Y register M=(Y)

Il contenuto del registro Y viene posto nella cella di memoria di indirizzo specificato. Il contenuto del registro Y non viene cambiato.
INDIRIZZAMENTO: assoluto — pagina 0 — pagina 0 indicizzato X.
FLAG MODIFICATI: nessuno.

TAX: Transfer Accumulator to X register X=(A)

Il contenuto dell'accumulatore viene posto nel registro X. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

TAY: Transfer Accumulator to Y register Y=(A)

Il contenuto dell'accumulatore viene posto nel registro Y. Il contenuto dell'accumulatore non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

TSX: Transfer Stack pointer to X register X=(S)

Il contenuto dello stack pointer viene posto nel registro X. Il contenuto dello stack pointer non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

TXA: Transfer X register to Accumulator A=(X)

Il contenuto del registro X viene posto nell'accumulatore. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

TXS: Transfer X register to Stack pointer S=(X)

Il contenuto del registro X viene posto nello stack pointer. Il contenuto del registro X non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

TYA: Transfer Y register to Accumulator A=(Y)

Il contenuto del registro Y viene posto nell'accumulatore. Il contenuto del registro Y non viene cambiato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

1.4.3 Operazioni logico matematiche

ADC: ADd with Carry A=(A)+DATO+C

Somma il contenuto dell'accumulatore con il dato specificato e il carry. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — indicizzato X e Y — pagina 0 indicizzato X — indiretto indicizzato — indicizzato indiretto.
FLAG MODIFICATI: carry, zero, overflow e segno.

AND: AND A=(A)∧DATO

Esegue l'AND logico tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — pagina 0 indicizzato X — indicizzato X e Y — indiretto indicizzato — indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

ASL: Arithmetic Shift Left C=B7=B6=B5=B4=B3=B2=B1=B0=0

Sposta il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso sinistra. Il contenuto del bit 0 diventa 0 e il contenuto del bit 7 viene posto nel flag di carry. Questa operazione equivale ad eseguire una moltiplicazione per 2.
INDIRIZZAMENTO: accumulatore — assoluto — pagina 0 — indicizzato X — pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

BIT: test BITs in memory (A)∧(M) — N=(M7) — V=(M6)

Esegue la AND logica tra il contenuto dell'accumulatore e il contenuto della cella di memoria specificata. Il risultato non viene posto da nessuna parte ma viene alterato il flag di zero. Pone il contenuto del bit 7 della cella di memoria specificata nel flag di segno e il contenuto del bit 6 nel flag di overflow.
INDIRIZZAMENTO: assoluto — pagina 0.
FLAG MODIFICATI: zero, overflow e segno.

CMP: CoMPare with accumulator (A)—DATO

Sottrae al contenuto dell'accumulatore il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — indicizzato X e Y — pagina 0 indicizzato X — indiretto indicizzato — indicizzato indiretto.
FLAG MODIFICATI: carry, zero e segno.

CPX: ComPare with X register (X)—DATO

Sottrae al contenuto del registro X il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).
INDIRIZZAMENTO: immediato — assoluto — pagina 0
FLAG MODIFICATI: carry, zero e segno.

CPY: ComPare with Y register (Y)—DATO

Sottrae al contenuto del registro Y il dato specificato. Il risultato non viene posto da nessuna parte ma vengono alterati i flag di carry, zero e segno (C=1 indica A>=DATO, Z=1 indica A=DATO).
INDIRIZZAMENTO: immediato — assoluto — pagina 0
FLAG MODIFICATI: carry, zero e segno.

DEC: DEcrement M=(M)—1

Decrementa il contenuto della cella di memoria specificata. Il risultato viene posto nella cella stessa.
INDIRIZZAMENTO: assoluto — pagina 0 — indicizzato X — pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

DEX: DEcrement X register X=(X)—1

Decrementa il contenuto del registro X. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

DEY: DEcrement Y register Y=(Y)—1

Decrementa il contenuto del registro Y. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

EOR: Exclusive OR A=(A)∀DATO

Esegue l'OR esclusivo tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — pagina 0 indicizzato X — indicizzato X e Y — indiretto indicizzato — indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

INC: INCrement M=(M)+1

Incrementa il contenuto della cella di memoria specificata. Il risultato viene posto nella cella stessa.
INDIRIZZAMENTO: assoluto — pagina 0 — indicizzato X — pagina 0 indicizzato X.
FLAG MODIFICATI: zero e segno.

INX: INcrement X register X=(X)+1

Incrementa il contenuto del registro X. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

INY: INcrement Y register Y=(Y)+1

Incrementa il contenuto del registro Y. Il risultato viene posto nello stesso registro.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: zero e segno.

LSR: Logic Shift Right C=B0=B1=B2=B3=B4=B5=B6=B7=0

Sposta il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso destra. Il contenuto del bit 7 diventa 0 e il contenuto del bit 0 viene posto nel flag di carry. Questa operazione equivale ad eseguire una divisione intera per 2.

INDIRIZZAMENTO: accumulatore — assoluto — pagina 0 — indicizzato X — pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

ORA: OR Accumulator A=(A)VDATO

Esegue l'OR logico tra l'accumulatore e il dato specificato. Il risultato viene posto nell'accumulatore.
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — indicizzato X e Y — pagina 0 indicizzato X — indiretto indicizzato — indicizzato indiretto.
FLAG MODIFICATI: zero e segno.

ROL: ROtate Left C=B7=B6=B5=B4=B3=B2=B1=B0=C

Ruota il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso sinistra. Nel bit 0 viene posto il contenuto del flag di carry, nel quale viene trasferito il contenuto del bit 7.
INDIRIZZAMENTO: accumulatore — assoluto — pagina 0 — indicizzato X — pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

ROR: ROtate Right C=B0=B1=B2=B3=B4=B5=B6=B7=C

Ruota il contenuto dell'accumulatore o della cella di memoria specificata di una posizione bit verso destra. Nel bit 7 viene posto il contenuto del flag di carry, nel quale viene trasferito il contenuto del bit 0.
INDIRIZZAMENTO: accumulatore — assoluto — pagina 0 — indicizzato X — pagina 0 indicizzato X.
FLAG MODIFICATI: carry, zero e segno.

SBC: SuBtract with Carry A=(A)—DATO—$\overline{C}$

Sottrae al contenuto dell'accumulatore il dato specificato. Il risultato viene posto nell'accumulatore. Devi usare il carry al contrario di come lo usi per l'istruzione ADC: carry a 0 vuol dire prestito; devi quindi porre a 1 il carry prima di eseguire un'operazione SBC senza prestito. Dopo un'operazione SBC si ha: C=1 indica A>=DATO, Z=1 indica A=DATO.
INDIRIZZAMENTO: immediato — assoluto — pagina 0 — indicizzato X e Y — pagina 0 indicizzato X — indiretto indicizzato — indicizzato indiretto.
FLAG MODIFICATI: carry, zero, overflow e segno.

1.4.4 Istruzioni di controllo del flusso

BCC: Branch on Carry Clear

Salta all'indirizzo specificato se il flag di carry contiene 0.
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BCS: Branch on Carry Set

Salta all'indirizzo specificato se il flag di carry contiene 1.
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BEQ: Branch on EQual

Salta all'indirizzo specificato se il flag di zero contiene 1 (cioè se il risultato è 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BMI: Branch on MInus

Salta all'indirizzo specificato se il flag di segno contiene 1 (cioè se il risultato è negativo).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BNE: Branch on Not Equal

Salta all'indirizzo specificato se il flag di zero contiene 0 (cioè se il risultato è diverso da 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BPL: Branch on PLus

Salta all'indirizzo specificato se il flag di segno contiene 0 (cioè se il risultato è positivo o uguale a 0).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BRK: BReaK

Salva nello stack (PC)+2 e i flag, salta all'indirizzo indicato da $FFFE $FFFF (come per gli interrupt). Per riconoscere se il salto è stato generato da un interrupt o dall'istruzione BRK devi guardare il flag di break salvato nello stack. BRK viene usato per trovare errori nel programma (come lo STOP del BASIC).
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: break.

BVC: Branch on oVerflow Clear

Salta all'indirizzo specificato se il flag di overflow contiene 0 (cioè se non si è verificato overflow).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

BVS: Branch on oVerflow Set

Salta all'indirizzo specificato se il flag di overflow contiene 1 (cioè se si è verificato overflow).
INDIRIZZAMENTO: relativo.
FLAG MODIFICATI: nessuno.

JMP: JuMP

Salta all'indirizzo specificato.
INDIRIZZAMENTO: assoluto — indiretto.
FLAG MODIFICATI: nessuno.

JSR: Jump SubRoutine

Salva nello stack (PC)+2 (l'indirizzo prima dell'istruzione che segue JSR), salta all'indirizzo specificato.
INDIRIZZAMENTO: assoluto.
FLAG MODIFICATI: nessuno.

NOP: No OPeration

Non fa nulla per due cicli di clock. Si usa per cicli di ritardo.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

RTI: ReTurn from Interrupt

Estrae dallo stack il registro di stato e il program counter che erano stati salvati da una chiamata ad interrupt o da una istruzione BRK.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: tutti.

RTS: ReTurn from Subroutine

Estrae dallo stack il program counter che era stato salvato da una istruzione JSR e lo incrementa di 1.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

1.4.5 Istruzioni sui flag

CLC: CLear Carry C=0

Viene azzerato il flag di carry. Generalmente si usa prima di un'istruzione ADC.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: carry.

CLD: CLear Decimal D=0

Viene azzerato il flag BCD. Quando questo flag contiene 0, l'ALU esegue le operazioni aritmetiche in binario.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: decimale.

CLI: CLear Interrupt I=0

Viene azzerato il flag di interrupt. Quando questo flag contiene 0 la CPU risponde alle chiamate d'interrupt.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: interrupt.

CLV: CLear oVerflow V=0

Viene azzerato il flag di overflow.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: overflow.

SEC: SEt Carry C=1

Viene posto a uno il flag di carry. Generalmente si usa prima un'istruzione SBC.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: carry.

SED: SEt Decimal D=1

Viene posto a uno il flag decimale. Quando questo flag contiene 1, l'ALU esegue le operazioni aritmetiche in BCD.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: decimale.

SEI: SEt Interrupt I=1

Viene posto a uno il flag di interrupt. Quando questo flag contiene 1 la CPU non risponde alle chiamate d'interrupt.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: interrupt.

1.4.6 Operazioni sullo stack

PHA: PusH Accumulator STACK=(A) ; SP=SP—1

Il contenuto dell'accumulatore viene memorizzato nella cella di indirizzo $100+(SP); lo stack pointer viene decrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

PHP: PusH Processor status STACK=(P) ; SP=SP—1

Il contenuto del registro di stato viene memorizzato nella cella di indirizzo $100+(SP); lo stack pointer viene decrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

PLA: PuL1 Accumulator A=(STACK) ; SP=SP+1

Il contenuto della cella di indirizzo $100+(SP) viene caricato nell'accumulatore; lo stack pointer viene incrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: nessuno.

PLP: PuL1 Processor status P=(STACK) ; SP=SP+1

Il contenuto della cella di indirizzo $100+(SP) viene caricato nel registro di stato; lo stack pointer viene incrementato.
INDIRIZZAMENTO: implicito.
FLAG MODIFICATI: tutti.

1.5 SCRITTURA DEI PROGRAMMI IN LINGUAGGIO MACCHINA

Programmare in linguaggio macchina non è molto diverso dal farlo in BASIC: il problema principale è che le istruzioni a disposizione sembrano molto meno potenti. Rimane comunque consigliabile affrontare i problemi come già visto nei volumi dedicati al BASIC, descrivendoli prima a grandi blocchi e sviluppando ognuno di questi blocchi in un secondo momento.

Le sequenze complesse di istruzioni che vanno eseguite in più punti del programma possono diventare sottoprogrammi. E possono essere sfruttate routine già scritte da altri programmatori, ad esempio quelle dell'Interprete BASIC o del KERNAL del Commodore 64.

Come per il BASIC, però, una volta risolto il problema e scritto il programma bisogna sapere come introdurlo in memoria, memorizzarlo e provarlo; scopo di questo paragrafo è spiegare come farlo.

1.5.1 Con PEEK e POKE

Il modo più elementare per scrivere nella memoria del COMMODORE 64 un programma in linguaggio macchina è quello di usare una serie di POKE con dati ed indirizzi prefissati ed utilizzare poi l'istruzione BASIC "SYS *indirizzo*" o la funzione "USR(*argomento*)" per attivare la routine. Il programmino che segue, ad esempio, memorizza ed esegue tre istruzioni in linguaggio macchina che scrivono una A nella prima posizione del video:

esadecimale	mnemonico
A9 41	LDA 'A
8D 00 04	STA $0400
60	RTS

```
5  REM A0
10 POKE 49152,169
20 POKE 49153, 65
30 POKE 49154,141
40 POKE 49155,  0
50 POKE 49156,  4
60 POKE 49157, 96
70 SYS 49152
```

Il secondo programma, invece, legge indirizzo di inizio e codici relativi ad una routine che trasforma da diretto a inverso e viceversa il codice dei primi 240 caratteri dello schermo:

```
5   REM A1
10  RESTORE
20  READ IND: XX=IND: READ YY
30  READ A$: IF A$="*" THEN 100
40  POKE IND,VAL(A$): IND=IND+1
50  GOTO 30
100 PRINT"INDIRIZZO DI INIZIO:"XX
110 PRINT"RICHIAMARE CON: SYS"YY
120 END
200 DATA 49152: REM INDIRIZZO DI INIZIO CODICE
210 DATA 49152: REM INDIRIZZO DI RICHIAMO
215 REM  DECIMALE       MNEMONICO/ASSEMBLER
220 DATA 162,240    :REM        LDX #F0
230 DATA 189,255,3  :REM CICLO: LDA $03FF,X
240 DATA 73,128     :REM        EOR #$80
250 DATA 157,255,3  :REM        STA $03FF,X
260 DATA 202        :REM        DEX
270 DATA 208,245    :REM        BNE CICLO
280 DATA 96         :REM        RTS
290 DATA *
```

Certo non è questo il modo più comodo per scrivere un programma ma può essere utile quando non si hanno strumenti più adatti.

A differenza dei suoi "cugini" del COMMODORE 16, PLUS 4 (3.5) e 128 (7.0), il BASIC del COMMODORE 64 (2.0) non dispone di un'istruzione MONITOR che permetta di accedere ad un mini assemblatore-disassemblatore che semplifichi il lavoro di scrittura di programmi in linguaggio macchina.
Esistono allora almeno due soluzioni: scriversi un piccolo monitor in BASIC oppure acquistare uno dei programmi tra le decine ormai disponibili per questo computer.

La prima soluzione va bene se vuoi provare con brevi sequenze di codici esadecimali a prendere confidenza con i problemi posti dal linguaggio macchina. Ecco, ad esempio, un programma BASIC che permette di leggere/modificare un byte alla volta la memoria del COMMODORE 64:

```
0   REM A2
1   REM MINI-MONITOR
3   :
5   HX$="0123456789ABCDEF"
90  GOTO 500
99  REM ---
100 IND=IND+1+65536*(IND=65535): RETURN
103 REM ---
105 IND=IND-1-65536*(IND=0): RETURN
108 REM ---
110 X$=MID$(HX$,X/16+1,1)+MID$(HX$,(X AND 15)+1,1): RETURN
118 REM ---
120 X=INT(IND/256): GOSUB 110: IND$=X$
125 X=IND- X*256: GOSUB 110:IND$=IND$+X$: RETURN
130 X=0:PT=1
135 IF LEN(X$)=0 THEN RETURN
140 X1$=MID$(X$,PT,1)
145 IF X1$>="0" AND X1$<="9" THEN X=X*16+ASC(X1$)-48: GOTO 160
150 IF X1$>="A" AND X1$<="F" THEN X=X*16+ASC(X1$)-55: GOTO 160
155 RETURN
160 PT=PT+1: GOTO 140
170 IF X$="" THEN PRINT"■■": GOSUB 400: RETURN
175 IF A$=" " THEN GOSUB 130: GOTO 185
180 X=VAL(X$)
185 POKE IND,X: PRINT"■■": GOSUB 400:RETURN
190 POKE 198,2:POKE 631,32:POKE 632,ASC(A$):RETURN
200 X$="":I=0:PRINT"■'A$"■";
205 PRINT"<■■";
210 GETB$:IFB$="" THEN 210
215 IF B$=CHR$(13) THEN RETURN
```

```
220 IF B$=CHR$(20) AND I>0 THEN I=I-1:X$=LEFT$(X$,I):PRINT" ■■■";:GOTO205
225 IF B$=CHR$(20) THEN 210
230 IF A$="/" THEN 250
235 IF I=2 AND A$=" " THEN 210
240 IF I=4 AND(A$="#" OR A$="↑")THEN210
245 IF B$>="A" AND B$<="F" THEN 255
250 IF B$<"0" OR B$>"9" THEN 210
255 IF A$="/" AND VAL(X$+B$)>255THEN210
260 X$=X$+B$:I=I+1:PRINT B$;:GOTO 205
300 GOSUB 200: GOSUB 130
305 IF X$<>"" THEN IND=X: GOTO 315
310 X=-1
315 RETURN
400 PRINT CHR$(13)"                                "CHR$(13)"□";
405 GOSUB 120: X=PEEK(IND): GOSUB 110:PRINT IND$" "X$;
410 PRINT SPC(7-LEN(STR$(IND)))IND"■■"X,
415 RETURN
500 GOSUB 400
505 GET A$: IF A$="" THEN 505
510 IF A$="■" THEN GOSUB 100: GOTO 500
515 IF A$="□" THEN GOSUB 105: GOTO 500
520 IF A$=" " THEN GOSUB 200: GOSUB 170: GOSUB 100: GOTO 500
525 IF A$="/" THEN GOSUB 200: GOSUB 170: GOSUB 100: GOTO 500
530 IF A$="↑" THEN GOSUB 300: IF X>=0 THEN SYS IND : GOTO 500
535 IF A$="↑" THEN 500
540 IF A$="#" THEN GOSUB 300: GOTO 500
550 IF A$>="0" AND A$<="9" OR A$>="A" AND A$<="F" THEN GOSUB 190: GOTO 505
590 IF A$="□" THEN PRINTA$;: GOTO 500
595 GOTO 505
```

Il programma propone l'indirizzo contenuto nella variabile IND ed il contenuto della locazione di memoria a quell'indirizzo prima in forma esadecimale e poi in forma decimale.

A questo punto attende la pressione di uno dei seguenti tasti:

CURSORE GIU' : incrementa di uno IND e torna alla fase di visualizzazione (se IND vale 65535 diventa 0);

CURSORE SU : decrementa di uno IND e torna alla fase di visualizzazione (se IND vale 0 diventa 65535);

/ : legge un numero decimale da sostituire al contenuto attuale del byte indirizzato da IND. Il dato può essere corretto usando il tasto DEL e terminato premendo RETURN. Per non effettuare modifiche basta premere RETURN dopo aver cancellato tutte le cifre eventualmente introdotte;

SPAZIO oppure
cifra tra 0 e 9 oppure
lettera tra 'a' e 'f': legge, con le stesse modalità del numero decimale, un valore esadecimale da memorizzare al posto di quello attuale.
Naturalmente, né in esadecimale né in decimale, è possibile fornire un dato il cui valore superi 255.
Dopo aver modificato la memoria, il valore di IND viene incrementato automaticamente;

: legge un valore esadecimale di 4 cifre da assegnare ad IND, serve per spostarsi velocemente nella zona di memoria su cui si vuole lavorare;

^ : legge un indirizzo esadecimale dove deve comparire una routine in linguaggio macchina terminante con l'istruzione RTS a cui cede il controllo.
Anche per questi due ultimi comandi vale che se il parametro fornito è nullo l'operazione non viene effettuata;

SHIFT+CLEAR/HOME : consente di pulire lo schermo.

1.5.2 I monitor

Un monitor completo, magari di quelli forniti in cartuccia assieme al tasto di RESET (utilissimo quando si lavora al di fuori dell'ambiente BASIC) ti sarà indispensabile se intendi utilizzare in modo estensivo il linguaggio macchina. I vari prodotti in circolazione utilizzano notazioni diverse per eseguire operazioni analoghe. Quello che segue è un breve schema delle operazioni normalmente disponibili.

Per ogni operazione indichiamo la sintassi dei comandi del MONITOR incluso nel sistema di sviluppo ASSEMBLER della Commodore, molto simile a quella dei comandi di HESMON, ZOOM ed altri. I parametri dei comandi sono numeri in formato esadecimale.

LETTURA/MODIFICA DELLA MEMORIA

C IND1 IND2 IND3 — confronta blocchi di memoria.
Confronta il blocco di memoria compreso tra gli indirizzi IND1 e IND2 con uno di eguali dimensioni e inizio in IND3. Vengono segnalati, in ordine crescente, tutti gli indirizzi corrispondenti il cui contenuto non sia uguale.

F IND1 IND2 BYTE — riempie una zona di memoria.
Riempie la zona di memoria compresa tra gli indirizzi IND1 ed IND2 (incluso) con il valore BYTE.

H IND1 IND2 DATI — cerca una sequenza di byte.
Cerca nella zona di memoria compresa tra gli indirizzi IND1 e IND2 la sequenza di byte indicata. Ogni volta che la trova ne visualizza l'indirizzo di inizio.
È possibile indicare la sequenza di byte o come sequenza di dati esadecimali, ad esempio:

H 1000 2000 41 42 43 44 45

oppure come stringa di caratteri:

H 1000 2000 'ABCDE

Il numero di byte specificabili e il delimitatore della stringa di caratteri (in questo esempio era l'apice singolo) dipende dal monitor.

I IND1 IND2 — visualizza la memoria in ASCII.
Mostra il contenuto della zona di memoria tra IND1 ed IND2 in forma di caratteri del set CBM-ASCII in campo inverso. I caratteri non stampabili sono visualizzati come '.'.

M IND1 IND2 — visualizza la memoria in esadecimale.

: INDI DATI — modifica la memoria in esadecimale.
Visualizza sullo schermo il contenuto esadecimale della memoria tra IND1 ed IND2 seguito dalla descrizione in CBM-ASCII con il seguente formato:

:zzzz x1 x2 x3 x4 x5 x6 x7 x8 yyyyyyyy

dove zzzz è l'indirizzo del primo degli otto byte, x1...x8 sono gli otto byte in esadecimale ed yyyyyyyy sono gli otto caratteri ASCII corrispondenti.
Se IND2 manca vengono visualizzati gli otto byte con inizio in IND1.

A questo punto è possibile modificare la memoria semplicemente spostandosi sui valori esadecimali che si vogliono alterare, modificandoli e premendo RETURN: la linea verrà riscritta aggiornata.

Normalmente, utilizzando i tasti CURSORE SU e GIU' dopo i comandi I ed M è possibile continuare l'operazione di lettura/modifica sugli indirizzi di memoria adiacenti a quelli visibili sullo schermo.

R — visualizza il contenuto dei registri della CPU 6510.

; DATI — modifica il contenuto dei registri della CPU.
Questo comando visualizza il contenuto dei registri al momento in cui è stato ceduto il controllo al MONITOR, con il seguente formato:

```
  PC    IRQ  SR AC XR YR SP
;XXXX YYYY AA BB CC DD EE
```

cioè con il valore esadecimale contenuto in ciascun registro, al momento in cui il controllo è passato al monitor, scritto sotto il nome mnemonico di quel registro.

Nei diversi monitor tali mnemonici o il loro ordine può differire leggermente. SR indica il registro dei FLAG (Status Register) mentre IRQ indica il contenuto dei byte $0314—$0315 (788-789) cioè l'indirizzo della routine di gestione delle interruzioni (vedi paragrafo 1.7) e non tutti i monitor lo visualizzano.

Per alterare il valore dei registri è sufficiente modificare quello visualizzato sulla linea che inizia con ';' sotto il nome del registro che interessa e premere RETURN: quando il controllo viene lasciato ad una routine in linguaggio macchina (vedi comandi G e W) i registri sono inizializzati con tali valori. Fa eccezione il 'registro' IRQ che viene modificato IMMEDIATAMENTE.

T IND1 IND2 IND3 — Trasferisce un blocco di memoria.
Copia il contenuto del blocco di memoria compreso tra gli indirizzi IND1 ed IND3 nel blocco che inizia all'indirizzo IND3.

Quasi tutti i monitor mettono poi a disposizione uno o più comandi che aiutano a correggere gli indirizzi di salto e degli operandi di un programma in linguaggio macchina che per qualche ragione debba essere spostato da una zona all'altra della memoria oppure che debba essere adattato al COMMODORE64 perché originariamente scritto per un altro modello di computer della COMMODORE. Ad esempio:

N IND1 IND2 IND3 IND4 IND5 — corregge gli indirizzi in un blocco di memoria.
Questo comando interpreta la memoria compresa tra IND1 ed IND2 come se dovesse disassemblarla (vedi comando D, più avanti) e se trova riferimenti ad indirizzi compresi tra IND4 ed IND5 li corregge aggiungendo IND3.

Se dopo IND5 aggiungiamo una 'W', invece, significa che la zona di memoria tra IND1 e IND2 non contiene codice macchina ma solo indirizzi (ad esempio una tabella di salti a routine) ed il monitor non deve quindi disassemblare il blocco di memoria per trovare gli indirizzi da controllare e correggere.

ASSEMBLA CODICE MACCHINA

A IND MNEM OPERANDO — assembla una linea di codice 6510.
Questo comando permette di assemblare istruzioni in linguaggio macchina.
IND è l'indirizzo a cui il codice viene assemblato; MNEM è il codice mnemonico dell'istruzione come indicato nel paragrafo 1.4;
OPERANDO, se presente, è il parametro dell'istruzione ed indica anche il modo di indirizzamento, va scritto come indicato nel paragrafo 1.3.
Premendo RETURN il monitor tenta di codificare l'istruzione: se è scritta in modo corretto il monitor la assembla e sulla linea seguente scrive: A *indirizzo-aggiornato* ed attende una nuova linea di codice.
Un carattere '?' indica invece che l'istruzione è errata o scritta male: utilizzando l'editor di schermo è possibile correggerla e proseguire nell'introduzione delle istruzioni.

DEBUGGING DEI PROGRAMMI IN LINGUAGGIO MACCHINA

D IND1 IND2 — visualizza la memoria in codice 6510 (disassembla).

,DATI — modifica la memoria in esadecimale.
Interpreta i byte tra IND1 e IND2 come istruzioni della CPU 6510. IND2 è opzionale e la mancanza di questo parametro genera il disassemblato di un numero di linee di codice dipendente dal tipo di monitor. La visualizzazione avviene nel formato:

,INDIRIZZO X1 X2 X3 MNEM OPERANDO

X1, X2 e X3 sono la traduzione in esadecimale dell'istruzione 6510, modificandoli e premendo RETURN la memoria viene ri-disassemblata e la linea riscritta. Si può anche scrivere una A sopra il carattere ',' e modificare direttamente il codice mnemonico e/o l'operando, come se si iniziasse ad assemblare, normalmente il monitor ignora i codici esadecimali.
I codici non riconosciuti vengono considerati come istruzioni da un solo byte con '???' come codice mnemonico; come per i comandi I ed M, può essere possibile continuare a disassemblare la memoria semplicemente utilizzando i tasti cursore per provocare lo scorrimento verticale dello schermo.

Tutti gli esempi di programmi dei prossimi paragrafi sono stati ottenuti utilizzando questo comando dopo il comando '0'.

Alcuni monitor consentono di disassemblare, in una zona di memoria, solo le istruzioni con un determinato modo di indirizzamento.

G IND — esegue un programma in linguaggio macchina.
Trasferisce il controllo al codice macchina che inizia all'indirizzo specificato o, se questo non è presente, al valore di PC ottenibile e modificabile con il comando R.

B IND CONT — seleziona un break-point.
Un breakpoint è l'equivalente in linguaggio macchina dell'istruzione STOP del BASIC. Utilizzando questo comando è possibile richiedere che l'esecuzione di un programma si interrompa prima che l'istruzione che inizia in IND sia eseguita. È così possibile controllare lo stato dei registri e della memoria. Il parametro CONT, quando è presente, indica che l'interruzione deve avvenire quando l'istruzione sta per essere eseguita la CONT-esima volta.
Non tutti i monitor ne permettono la definizione ed anche i dettagli nell'uso possono variare in modo significativo.

W IND — esegue un'istruzione per volta.
Questo comando esegue SOLO l'istruzione all'indirizzo specificato oppure, se tale parametro manca, all'indirizzo mostrato in PC dal comando R. Dopo di ciò il controllo ritorna al MONITOR che, in genere, visualizza il contenuto dei registri. In alcuni monitor il comando analogo è S, dall'inglese single-step = un-solo-passo.

Spesso i diversi programmi mettono a disposizione altri comandi di ausilio per la correzione dei programmi in linguaggio macchina. Uno dei più diffusi è TRACE, = traccia, che esegue un'istruzione, visualizza in contenuto dei registri e, se non si preme STOP, passa all'istruzione successiva. In Hesmon questo comando è attivato da Q (quicktrace).

INPUT-OUTPUT

L "nomefile" DN — legge un programma.
Questo comando è analogo a:

LOAD "nomefile",DN,1

dato in ambiente BASIC in modo immediato, salvo che DN è un numero esadecimale composto da due cifre (01 per il nastro, 08 per il disco, etc.).

L'effetto è quello di caricare nella memoria un programma in linguaggio macchina salvato in precedenza, riscrivendolo negli stessi indirizzi che occupava al momento del salvataggio.
Se DN manca si assume il nastro come unità di input, e se manca anche il nome del programma, viene caricato il primo che si incontra nella lettura.
Alcuni monitor consentono di indicare in quale indirizzo di memoria iniziare a caricare il programma, ignorando le informazioni contenute all'inizio del file (vedi comando 'S', di seguito).

S "nomefile" DN IND1 IND2 — salva un programma.
Questo comando permette di trascrivere nella memoria di massa indicata da DN un programma o comunque un blocco di memoria con inizio in IND1 e fine in IND2—1. L'unico parametro che può essere omesso, se DN è 01, è il nome del file.
Il file programma creato contiene, nei primi due byte, l'indirizzo IND1 che il comando 'L' utilizza per stabilire a quale indirizzo iniziare a caricare i dati.

O DN SA "nomefile" — seleziona il dispositivo di Output.
Con questo comando, non disponibile in tutti i monitor, è possibile modificare i dispositivo di Output per poter stampare o addirittura memorizzare in forma leggibile su disco o cassetta il risultato del proprio lavoro.
Tutti i parametri sono opzionali, 'O' da solo seleziona il video.
"nomefile" diventa obbligatorio se DN indica un'unità a dischi e in questo caso deve contenere anche le indicazioni sul tipo di file (Seq, Prg) ed il modo di accesso (W=scrittura, A=aggiunta).
Se il vostro monitor non dispone di questo comando potete provare ad aprire il file da basic con i comandi:

```
OPEN XX,DN,SA,"nomefile"
CMD  XX
```

Rientrando nel monitor, se questo non resetta le comunicazioni al momento dell'attivazione, il dispositivo di uscita sarà ancora quello a cui avrete riservato il numero logico XX.

RITORNO AL BASIC

X — torna al BASIC.
Per quanto possa sembrare strano, non tutti i monitor dispongono di questo comando che consente di ritornare al BASIC senza cancellare un eventuale programma BASIC presente in memoria prima dell'attivazione del monitor.
Una seconda forma del comando 'XC' permette di effettuare volontariamente un COLD-START del BASIC, necessario in alcuni monitor per rimettere a posto i puntatori in pagina 0 la prima volta che si rientra in ambiente BASIC.

1.5.3 Gli assemblatori

Se intendi scrivere PROGRAMMI in codice macchina lunghi e complessi ti conviene procurarti un assemblatore (ASSEMBLER), un programma che consente, oltre all'uso di nomi mnemonici al posto dei codici esadecimali delle istruzioni, anche l'uso di nomi per i parametri delle istruzioni o per gli indirizzi di salto (istruzioni BRANCH e JUMP) e la definizione di macro-istruzioni (sequenze di istruzioni in linguaggio macchina identificate con un nome simbolico che è poi utilizzato scrivendo il programma come se fosse una nuova istruzione della CPU 6510, i dettagli li trovi nei manuali d'uso dei diversi programmi. Altre caratteristiche sono le DIRETTIVE, pseudo-istruzioni che consentono di indicare al programma assemblatore di definire tabelle di indirizzi oppure inframmezzare il codice con la codifica ASCII di una stringa di caratteri, o ancora assemblare o meno un pezzo di programma se una certa costante è stata definita o ha un determinato valore.

Assieme all'assemblatore viene di solito fornito anche un editor, un programma per scrivere testi con il quale preparare il programma SORGENTE, cioè quello composto da istruzioni assembler, dichiarazioni di costanti e direttive. Il programma assemblatore legge il programma SORGENTE e genera il CODICE OGGETTO.

Poiché esiste la possibilità che il programma SORGENTE descriva zone di memoria non contigue (indirizzi non tutti uno successivo all'altro) è necessario utilizzare un apposito caricatore (LOADER), anch'esso incluso nei programmi forniti insieme all'assemblatore, per leggere in memoria il CODICE OGGETTO; questo infatti è scritto in modo tale da contenere, oltre al puro codice 6510, anche indicazioni sugli indirizzi ove tale codice va caricato e byte di controllo (CHECKSUM) per controllare che il caricamento sia stato correttamente effettuato.

1.6 INTERFACCIA CON IL BASIC

Abbiamo visto come introdurre nella memoria un programma in linguaggio macchina. Se non comprende un caricatore, per salvarlo su supporto magnetico da MONITOR è sufficiente utilizzare il comando 'S' (o uno equivalente). Da BASIC si può sfruttare la routine KERNAL che salva i programmi, ma questo lo vedremo più avanti.
Per rileggerlo in memoria si può utilizzare la routine di lettura del KERNAL direttamente o indirettamente, tramite il comando BASIC 'LOAD'. La difficoltà sorge dal fatto che questo comando esegue automaticamente anche il comando GOTO alla prima istruzione del programma.
Il trucco per utilizzare comunque questa istruzione da programma è di inizializzare una determinata variabile solo quando il caricamento è stato effettuato e di controllarne il valore in una delle prime linee di programma.

Ad esempio, se va caricata una sola routine:

```
10 IF A=1 THEN GOTO 30
20 A=1: LOAD "nome-programma",8,1
30 ...seguito del programma...
```

se le routine da caricare sono più di una il sistema funziona in maniera analoga:

```
10 IF A>0 THEN GOTO 30
20 A=1: LOAD "prima-parte",8,1
30 IF A=2 THEN GOTO 50
40 A=2: LOAD "seconda-parte",8,1
50 ...seguito del programma...
```

Una volta letto in memoria il programma in linguaggio macchina per eseguirlo, a meno che non si sia in ambiente MONITOR, è necessario utilizzare gli strumenti messi a nostra disposizione dal BASIC.

L'istruzione:

SYS *indirizzo*

trasferisce il controllo alla routine in linguaggio macchina che inizia alla locazione *indirizzo*, un numero decimale la cui parte intera è compresa tra 0 e 65535, e che per fare ritorno all'istruzione successiva alla SYS deve terminare con l'istruzione RTS. Viene infatti eseguita un'istruzione JMP ma solo dopo che nello stack è stato inserito l'indirizzo della parte finale della routine che esegue il comando SYS ed i registri della CPU sono stati caricati con i valori contenuti agli indirizzi sotto indicati:

$030C (780) Accumulatore
$030D (781) registro X
$030E (782) registro Y
$030F (783) registro dei FLAG.

Un utilizzo immediato di queste informazioni è ad esempio quello di "mimare" l'istruzione AT(riga,colonna) mancante nel BASIC 2.0. Come spiegato nel paragrafo sulle routine del KERNAL esiste già una routine che effettua il posizionamento del cursore sullo schermo. Da BASIC è richiamabile con:

```
POKE 781,riga:POKE 782,colonna:POKE783,0:SYS 65520
```

oppure:

POKE 781,riga:POKE 782,colonna:SYS 58636
(by-passando la tabella del KERNAL)

La routine può prevedere che altri parametri le siano passati in determinate locazioni della memoria oppure, come vedremo più avanti, prelevare questi dati direttamente dal testo del programma BASIC.
Al termine dell'esecuzione i registri della CPU vengono salvati nelle stesse locazioni viste prima e sono quindi leggibili con delle semplici PEEK.
Se vuoi che la routine in linguaggio macchina esegua semplicemente dei calcoli, magari più in fretta del BASIC, e ti restituisca almeno un valore, allora ti può essere utile la funzione:

variabile = USR(*argomento*)

che accetta un argomento numerico, lo trascrive nell'accumulatore in virgola mobile #1 con il formato:

$0061 (97) esponente
$0062 (98) mantissa
$0063 (99) mantissa
$0064 (100) mantissa
$0065 (101) mantissa
$0066 (102) segno

e quindi effettua un salto all'indirizzo scritto nei byte 785 e 786 ($0311-0312 byte basso-byte alto), utilizzando l'istruzione JMP il cui codice ($4C) è contenuto nel byte 784 ($0310). La routine all'indirizzo specificato può leggere il valore del parametro passato, effettuare le elaborazioni previste e trascrivere nell'accumulatore in virgola mobile l'eventuale risultato che viene restituito come valore della funzione.
L'unico problema è che se hai bisogno di calcolare più di una funzione devi aggiornare ogni volta l'indirizzo in 785-786.

Come indicato nella "Guida di riferimento per il programmatore" esistono almeno altri tre modi per attivare delle routine in linguaggio macchina.
Se controlli la mappa della memoria nel volume dedicato al BASIC puoi notare che a partire dall'indirizzo 768 ($0300) ci sono delle coppie di byte descritte come "Vettore.....". Ciascuna di queste coppie contiene l'indirizzo della routine che assolve ad un determinato compito. Modificando tale indirizzo è possibile sostituire completamente tale routine oppure aggiungervi in testa del codice che compia qualche operazione in più.
Questa modifica può essere fatta utilizzando una piccola routine in linguaggio macchina oppure una coppia di POKE, come nell'esempio che segue.

Scrivi questo piccolo programma:

0 REM
10 PRINT CHR$(160+RND(0)*32);

quindi, in modo immediato:

POKE770,125: POKE771,168: POKE768,125:POKE769,168 e poi premete RETURN.

Il programma si attiverà automaticamente ed ogni tentativo di fermarlo, a meno che tu non disponga di un tasto di RESET, non farà altro che farlo ricominciare da capo (con le POKE abbiamo alterato il vettore di WARM-START e quello di stampa dei messaggi di errore in modo che puntino alla routine che esegue il comando RUN).

Allo stesso modo si può sfruttare il fatto che se si esegue l'istruzione:

LOAD"programma",DN,1

il programma viene caricato all'indirizzo specificato nei suoi primi due byte. Per spiegarci meglio, vediamo cosa succede eseguendo il programma che segue:

```
5  REM A3
10 OPEN 2,8,2,"PROVA1,P,W"
20 PRINT#2,CHR$(0)CHR$(4);
30 FOR K=0TO999
40 PRINT#2,CHR$(112+RND(0)*4);
50 NEXT
60 CLOSE2
```

Viene creato un file programma di nome PROVA1 che contiene nei primi due byte l'indirizzo di inizio dello schermo.
Prova ora ad eseguire il comando:

LOAD"PROVA1",8,1, (o ,1,1 se hai usato il nastro).

Lo schermo si riempirà di caratteri grafici, quelli memorizzati nel file dal programma precedente.

Con questo meccanismo (lo stesso utilizzato dai comandi 'L' ed 'S' dei monitor) è possibile scrivere dei programmi in BASIC o in linguaggio macchina che partono automaticamente appena il caricamento ha termine. È sufficiente che l'indirizzo di inizio del caricamento consenta ai byte del programma di sostituirsi ai valori nei vettori di salto presenti in memoria dall'indirizzo 768 in poi.

Sempre nelle prime pagine della memoria è presente la routine che riconosce un carattere o un TOKEN nel testo BASIC. Il fatto che sia in RAM ne permette la modifica e quindi l'aggiunta di nuovi comandi al BASIC residente. Se però questi comandi devono poter comparire (ed essere listati) nei programmi è necessario anche modificare le routine di stampa dei TOKEN, valutazione delle funzioni (se i nuovi TOKEN si riferiscono anche a funzioni) e compressione in TOKEN di parti della linea di testo BASIC. Tutto ciò è fattibile poiché tali routine sono "vettorizzate", sempre nella tabella che inizia in 768.
In questa tabella compare anche l'indirizzo della routine che viene eseguita dalla CPU 6510 ogni 60esimo di secondo, routine che si occupa della gestione delle INTERRUZIONI.

1.7 LA GESTIONE DELLE INTERRUZIONI

La CPU 6510 dispone di un piedino da cui riceve le chiamate di INTERRUPT (=interruzione). Puoi immaginare un segnale di interruzione come la chiamata HARDWARE di un sottoprogramma; la subroutine cioè non viene eseguita in risposta ad una istruzione contenuta in un programma ma a un segnale logico che giunge ad un piedino della CPU.

Nella versione base del Commodore 64 gli unici dispositivi collegati alla linea di richiesta delle interruzioni sono il CIA 6526 (Complex Interface Adapter) ed il VIC II 6566 (quest'ultimo viene trattato esaustivamente nel volume dedicato alla grafica!); ogni 60esimo di secondo la CPU riceve da questo chip una richiesta di interruzione. Il SISTEMA OPERATIVO usa questo segnale per eseguire ad intervalli regolari alcune operazioni quali aggiornare TI, controllare la pressione del tasto STOP e memorizzare eventuali caratteri premuti sulla tastiera.

L'esempio che segue, riportato sia in esadecimale che in ASSEMBLER, mostra come la routine di gestione delle interruzioni, che può essere disabilitata, controlli lo stato della tastiera. Puoi introdurlo in memoria con uno dei metodi visti nel paragrafo 1.5:

```
C000 58        CLI        ; ABILITA LE INTERRUZIONI
C001 A5 C5     LDA $C5    ; IL CODICE DEL TASTO PREMUTO
C003 8D 00 04  STA $0400  ; VIENE TRASCRITTO NEL VIDEO
C006 18        CLC
C006 90 F8     BCC $C001  ; RIPETE ALL'INFINITO
```

Vedrai che il primo carattere in alto a sinistra nello schermo cambia in funzione del tasto che premi sulla tastiera; questo fatto parrebbe strano sapendo che la cella di indirizzo $C5 è una normale cella di memoria in pagina 0 e non un registro di ingresso.
Se arresti l'esecuzione del programma (con STOP+RESTORE) e cambi l'istruzione in $C000, cioè sostituisci CLI con SEI (codice $78, maschera le interruzioni), ti accorgi, avviando nuovamente il programma, che la tastiera non influisce più sul primo carattere del video poiché la ROUTINE DI GESTIONE DELLE INTERRUZIONI non viene eseguita quando il FLAG di INTERRUPT è a 1.

Questa routine può essere intercettata in modo da far eseguire ad intervalli regolari operazioni da noi predisposte oltre a quelle del SISTEMA OPERATIVO od in sostituzione ad esse. Nell'ultimo paragrafo di questo capitolo riportiamo un esempio di uso di questa caratteristica.

Quando la CPU risponde a una chiamata di interrupt, salva automaticamente nello STACK i seguenti dati:

- gli 8 bit più significativi del PC,
- gli 8 bit meno significativi del PC,
- il registro dei FLAG

ed effettua un salto indiretto a $FFFE.

È cura del SISTEMA OPERATIVO effettuare tutte le operazioni che si devono eseguire in risposta a un interrupt. Per terminare la routine di gestione delle interruzioni esiste l'istruzione RTI, diversa da RTS perché recupera dallo STACK anche il registro dei FLAG, oltre a non incrementare il PC.

1.8 IL SISTEMA OPERATIVO DEL CBM 64

All'accensione, il Commodore 64 entra automaticamente in ambiente BASIC a meno che all'indirizzo 32768 ($8000) non compaiano dati che indichino la presenza di una cartuccia nello slot di espansione e l'indirizzo di inizio del programma a cui cedere il controllo al posto dell'Interprete BASIC.

Quando si scrivono programmi in codice macchina o in assembler, la maggior parte delle difficoltà nasce da operazioni complicate come calcoli matematici (specialmente quelli in virgola mobile) e trattamento delle operazioni di Input/Output. Un altro problema è quello di adattare un programma in linguaggio macchina scritto originariamente per una macchina diversa o per una diversa versione della stessa macchina.

I programmatori della Commodore hanno parzialmente risolto questi problemi: molte delle routine matematiche e di I/O di cui un programmatore può avere bisogno sono già contenute nelle ROM del BASIC e del KERNAL, basta conoscerne l'indirizzo di inizio ed i parametri da passare. Anche il problema della compatibilità tra versioni o macchine diverse è stato risolto: tutti i computer COMMODORE, dai PET (serie 2000, 4000, 8000, etc) fino al CBM128 contengono in una zona prestabilita della ROM una tabella di salti all'insieme di routine che compongono il nucleo del Sistema Operativo di quel modello; naturalmente l'indirizzo della routine può cambiare da una versione all'altra ma l'istruzione di salto a quella routine è sempre nella stessa posizione della tabella. Così le modifiche da effettuare per aggiornare i programmi si riducono drasticamente.

1.8.1 Il KERNAL

In questo paragrafo riportiamo la tabella dei salti ed una breve spiegazione delle routine del KERNAL presenti nel Commodore 64.

NOME	IND.ESA.	IND.DEC.	FUNZIONE
ACPTR	FFA5	65445	Legge un byte dalla porta seriale
CHKIN	FFC6	65478	Apre un canale in input
CHKOUT	FFC9	65481	Apre un canale in output
CHRIN	FFCF	65487	Legge un carattere dal canale di input
CHROUT	FFD2	65490	Scrive un carattere sul canale di output
CINT	FF81	65409	Inizializza screen editor e VIC II
CIOUT	FFA8	65448	Invia un byt sul bus seriale
CLALL	FFE7	65511	Chiude tutti i canali e file aperti
CLOSE	FFC3	65475	Chiude un file logico
CLRCHN	FFCC	65484	Chiude tutti i canali di Input/Output
GETIN	FFE4	65508	Legge una carattere dal buffer di tastiera
IOBASE	FFF3	65523	Fornisce l'indirizzo della memoria in cui sono mappati i dispositivi di Input/Output
IOINIT	FF84	65412	Inizializza l'Input/Output
LISTEN	FFB1	65457	Predispone una periferica a ricevere dati dal bus seriale
LOAD	FFD5	65493	Legge RAM da una periferica
MEMBOT	FF9C	65436	Legge/scrive l'indirizzo di inizio RAM
MEMTOP	FF99	65433	Legge/scrive l'indirizzo di fine RAM
OPEN	FFC0	65472	Apre un file logico
PLOT	FFF0	65520	Legge/seleziona la posizione del cursore
RAMTAS	FF87	65415	Inizializza la RAM, buffer nastro e schermo
RDTIM	FFDE	65502	Legge l'orologio interno
READST	FFB7	65463	Legge la parola di stato
RESTOR	FF8A	65418	Ripristina i valori dei vettori di I/O
SAVE	FFD8	65496	Salva RAM su una periferica
SCNKEY	FF9F	65439	Scandisce la tastiera
SCREEN	FFED	65517	Fornisce il formato del video
SECOND	FF93	65427	Invia l'indirizzo secondario dopo LISTEN
SETLFS	FFBA	65466	Prepara i parametri per un file logico
SETMSG	FF90	65424	Controlla la stampa dei messaggi del KERNAL
SETNAM	FFBD	65469	Prepara il nome del file
SETTIM	FFDB	65499	Inizializza l'orologio interno
SETTMO	FFA2	65442	Seleziona il flag di timeout (solo IEEE)
STOP	FFE1	65505	Controlla la pressione del tasto STOP
TALK	FFB4	65460	Predispone una periferica a trasmettere dati sul bus seriale
TKSA	FF96	65430	Invia l'indirizzo secondario dopo TALK
UDTIM	FFEA	65514	Aggiorna l'orologio interno
UNLSN	FFAE	65454	Invia un comando UNLISTEN sul bus seriale
UNTLK	FFAB	65451	Invia il comando UNTALK sul bus seriale
VECTOR	FF8D	65421	Aggiorna i vettori di sistema in RAM

ACPTR: legge un byte dal bus seriale.
Chiamata: $FFA5 (65445)
Comunic.: reg. A
Preparazione: TALK, TKSA
Errore: READST
Byte stack: 13
Reg. usati: A e X
Restituisce in A un byte letto dal dispositivo a cui sono stati inviati il segnale di TALK e, se necessario (con TKSA), l'indirizzo secondario.

CHKIN: apre un canale per input.
Chiamata: $FFC6 (65478)
Comunic.: reg. X
Preparazione: OPEN
Errori: 3,5,6
Reg. usati: A e X
Definisce un file logico già aperto come un canale di input. Si deve usare prima di CHRIN o di GETIN. Per ricevere dalla tastiera, se non è aperto alcun altro canale di input, si può evitare di chiamare sia la OPEN che questa routine. Se viene usata con il bus seriale, essa invia automaticamente l'indirizzo di TALK, e anche SA, se specificato nella OPEN. Il numero LFN deve essere scritto nel registro X prima della chiamata.

CHKOUT: apre un canale per output.
Chiamata: $FFC9 (65481)
Comunic.: reg. X
Preparazione: OPEN
Errori: 3,5,7
Reg. usati: A e X
Definisce un file logico già aperto come un canale di output. Non si deve usare se l'output è diretto al video e non sono aperti altri canali di output (come anche OPEN). Se usata con il bus seriale, essa invia automaticamente l'indirizzo di LISTEN specificato nella OPEN, e anche SA, se è presente, LFN deve essere precedentemente scritto in X.

CHRIN: legge un carattere dal canale di input.
Chiamata: $FFCF (65487)
Comunic.: reg. A
Preparazione: OPEN, CHKIN
Errore: READST
Reg. usati: A e X
Deve essere usata dopo OPEN e CHKIN, a meno che non legga da tastiera. Il byte si trova in A. Il canale rimane aperto. Il comportamento per l'input da tastiera è il

seguente: la linea scritta va nel buffer quando si preme il RETURN. Si chiama la routine per prelevare dal buffer un carattere per volta controllando se si arriva al RETURN.

CHROUT: scrive un carattere.
Chiamata: $FFD2 (65490)
Comunic.: reg. A
Preparazione: OPEN, CHKOUT
Errore: READST
Manda un carattere su un canale già aperto. Si deve usare prima OPEN e CHKOUT; se non è così il carattere viene inviato al dispositivo 3, il video. Il carattere va posto in A. Il canale resta aperto. Se sul bus seriale sono aperti più dispositivi, il dato viene inviato a tutti; per questo bisogna lasciare aperto solo quello che interessa e chiudere gli altri.

CINT: inizializza screen-editor e chip 6567 (VIC II).
Chiamata: $FF81 (65409)
Byte stack: 4
Reg. usati: A, X e Y
Questa routine inizializza il controllore video del Commodore 64, il chip 6567, per operare in modo normale. Anche i parametri dell'editor di schermo sono inizializzati. La routine è stata aggiunta a quelle già esistenti nel KERNAL per poter essere richiamata dai programmi in cartuccia che vengono attivati PRIMA che lo schermo del Commodore 64 sia inizializzato.

CIOUT: invia un byte sul bus seriale.
Chiamata: $FFA8 (65448)
Comunic.: reg. A
Preparazione: LISTEN, SECOND
Errore: READST
Il dispositivo deve essere in stato LISTEN altrimenti si ha errore di timeout. Se necessario SA, deve essere usata prima la SECOND. Il carattere da inviare deve essere posto in A. La routine mantiene un carattere nel buffer. Quando viene chiamata UNLSN viene inviato il carattere che sta nel buffer insieme a EOI e il dispositivo esce dallo stato di attesa.

CLALL: chiude tutti i file.
Chiamata: $FFE7 (65511)
Byte stack: 11
Reg. usati: A e X
Viene resettato il puntatore alla tabella dei file aperti chiudendo tutti i file e resettando tutti i canali di I/O.

CLOSE: chiude un file logico.
Chiamata: $FFC3 (65475)
Comunic.: A
Errori: 0,240
Reg. usati: A e X
Si deve caricare in A il LFN del file da chiudere e chiamare la routine.

CLRCHN: chiude tutti i canali e rimette i canali di I/O ai valori normali (0 per la tastiera e 3 per il video).
Chiamata: $FFCC (65484)
Byte stack: 9
Reg. usati: A e X
Se un canale da chiudere riguarda la porta seriale, viene mandato o un UNTALK per input o un UNLISTEN per output. Si ricordi che se sulla porta seriale sono attivi più dispositivi, essi ricevono contemporaneamente dati.

GETIN: preleva un carattere dal buffer tastiera.
Chiamata: $FFE4 (65508)
Comunic.: A
Errori: READST
Reg. usati: A e X
Il carattere prelevato dal buffer si ritrova in A in codice ASCII. Se non ci sono caratteri A contiene zero. I caratteri sono messi nel buffer dalla routine SCNKEY che va in onda per effetto dell'interrupt 60 volte al sec.; il buffer può contenere 10 caratteri, quelli eccedenti vanno persi.

IOBASE: fornisce l'indirizzo della pagina in cui sono mappati gli indirizzi di I/O.
Chiamata: $FFF3 (65523)
Comunic.: X e Y
Byte stack: 2
Reg. usati: X e Y
Dopo la chiamata X contiene il byte basso e Y il byte alto dell'indirizzo. Usando questo indirizzo più un offset (spostamento) si può accedere alle locazioni che interessano per una periferica.
Questa routine è stata scritta, come altre nel KERNAL, per mantenere la compatibilità tra modelli e versioni diverse dei computer Commodore.

IOINIT: inizializza i dispositivi di I/O.
Chiamata: $FF84 (65412)
Reg. usati: A, X e Y
Questa routine inizializza tutti i dispositivi e le routine di input/output. Come CINT è normalmente utilizzata dai programmi su cartuccia (cartridge).

LISTEN: predispone una periferica a ricevere dati dal bus seriale.
Chiamata: $FFB1 (65457)
Comunic.: A
Errori: READST
Reg. usati: A
Prima della chiamata si deve porre in A un numero tra 4 e 30; esso viene decodificato per ottenere l'indirizzo della periferica; questo le viene inviato e la pone in stato di attesa.

LOAD: trasferisce byte da una periferica alla memoria del COMMODORE 64.
Chiamata: $FFD5 (65493)
Comunic.: A, X, Y
Preparazione: SETLFS, SETNAM
Reg. usati: A, X e Y
Errori: 4,5,8,9
Prima della chiamata si devono usare SETLFS e SETNAM e caricare A con 0 per LOAD o con 1 per VERIFY. Se il file è stato aperto con SA = 0 (rilocazione) vengono ignorate le informazioni di testata del file, ma i registri X e Y devono contenere l'indirizzo di inizio memorizzazione. Se invece SA è 1, 2 o 3 l'indirizzo di inizio memorizzazione viene prelevato dalla testata del file. Dopo l'esecuzione X e Y contengono l'indirizzo dell'ultima locazione RAM caricata.
Questa routine non va utilizzata per leggere da tastiera, video o RS-232.

MEMBOT: definisce l'inizio della memoria RAM.
Chiamata: $FF9C (65436)
Comunic.: X e Y
Reg. usati: X e Y
Byte stack: 2
Può essere usata in 2 modi: ponendo il bit di Carry a 1 si ha in X e Y l'indirizzo di inizio RAM, cioè una lettura; ponendo il bit di Carry a 0 si scrive nei puntatori all'inizio RAM il valore contenuto in X e Y (X byte basso e Y byte alto).

MEMTOP: definisce la fine della memoria.
Chiamata: $FF99 (65433)
Comunic.: X e Y
Reg. usati: X e Y
Byte stack: 2
Può essere usata in 2 modi: con Carry a 1 si legge in X e Y l'indirizzo di fine memoria; con Carry a 0 si scrive nei puntatori alla fine della memoria e il contenuto dei registri X e Y. In questa seconda forma si usa per lasciare della memoria libera agli indirizzi alti per programmi in linguaggio macchina.

OPEN: apre un file logico.
Chiamata: $FFC0 (65472)
Reg. usati: A, X e Y
Preparazione: SETLFS, SETNAM
Errori: 1,2,4,5,8,240
La routine non ha parametri e lavora su quelli sistemati dalle due routine che devono essere chiamate prima: SETNAM e SETLFS. Dopo questa routine si può lavorare su un file logico che risulta completamente definito.

PLOT: riguarda la posizione del cursore sul video.
Chiamata: $FFF0 (65520)
Comunic.: A, X e Y
Byte stack: 2
Reg. usati: A, X e Y
Può essere usata in due modi. Il reg. X si riferisce alla riga (0-24), Y alla colonna (0-39). Se Carry=1 legge in X e Y la posizione corrente del cursore, se Carry=0 scrive, cioè sposta il cursore nella posizione determinata da X e Y.

RAMTAS: esegue un test sulla memoria RAM.
Chiamata: $FF87 (65415)
Comunic.: A, X e Y
Byte stack: 2
Reg. usati: A, X e Y
Questa routine determina dove termina la RAM dedicata all'utente e seleziona di conseguenza inizio e fine della memoria. Azzera inoltre le locazioni da $0000 a $0101 (257) e da $0200 (512) a $03FF (1023), alloca il buffer per l'unità a nastro e lo schermo video in $0400 (1024).

RDTIM: legge il contenuto dell'orologio.
Chiamata: $FFDE (65502)
Comunic.: A, X e Y
Byte stack: 2
Reg. usati: A, X e Y
L'orologio è un contatore che occupa 3 byte e viene aggiornato 60 volte al secondo. All'uscita si ha: byte più significativo in A, successivo in X e meno significativo in Y.

READST: legge la parola di stato.
Chiamata: $FFB7 (65463)
Comunic.: A
Byte stack: 2
Reg. usati: A
Trasferisce in A la parola di stato, chiamandola dopo un'operazione di I/O si sa se ci sono stati errori.

I bit di ST hanno il significato seguente:

Posiz. del bit in ST	Valore di ST	Lettura da nastro	Input/Output seriale	Verify/Load da nastro
0	1		timeout in scrittura	
1	2		timeout in lettura	
2	4	Blocco corto		Blocco corto
3	8	Blocco lungo		Blocco lungo
4	16	Errore fatale		I dati non coincidono
5	32	Errore di checksum		Errore di checksum
6	64	Fine file	EOI	
7	—128	Fine nastro	Disp. non presente	Fine nastro

RESTORE: ripristina i valori iniziali in tutti i vettori di interrupt usati dal sistema.
Chiamata: $FF8A (65418)
Byte stack: 2
Reg. usati: A, X e Y
Questa routine ripristina tutti i valori dei vettori di salto utilizzati per gestire gli INTERRUPT e le altre routine del KERNAL e del BASIC.

SAVE: salva una parte di memoria su una periferica.
Chiamata: $FFD8 (65496)
Comunic.: A, X e Y
Preparazione: SETLFS, SETNAM
Reg. usati: A, X e Y
Errori: 5,8,9
Copia un pezzo di memoria il cui indirizzo di inizio in pagina zero è puntato dal contenuto di A, mentre quello di fine è puntato dal contenuto di X e Y. Prima si devono usare SETNAM e SETLFS per definire il file di memorizzazione. Il nome del file può essere omesso se il salvataggio viene effettuato su cassetta. Non può essere usato come numero di dispositivo 0 (tastiera) e 3 (video). Prima di chiamare la routine si deve scrivere, in due locazioni contigue della pagina zero, l'indirizzo di inizio della memoria da salvare e porre in A il puntatore alla prima di queste due locazioni. Si ricordi che gli indirizzi si scrivono anteponendo il byte basso a quello alto.

SCNKEY: scandisce la tastiera.
Chiamata: $FF9F (65439)
Preparazione: IOINIT
Reg. usati: A, X e Y
Scandisce la tastiera e pone nel buffer il codice ASCII del carattere che corrisponde al tasto premuto. Il buffer della tastiera va da 631 a 640 (0277H-0280H). Viene chiamata 60 volte al secondo dalla routine di interrupt.

SCREEN: fornisce il formato del video.
Chiamata: $FFED (65517)
Comunic.: X e Y
Byte stack: 2
Reg. usati: X e Y
Il numero delle righe del video (25) va in Y e il numero di colonne (40) va in X. Può essere usata per adattare programmi alla configurazione del video del COMMODORE 64.

SECOND: invia l'indirizzo secondario SA.
Chiamata: $FF93 (65427)
Comunic.: A
Preparazione: LISTEN
Errore: READST
Reg. usati: A
Deve essere usata dopo LISTEN se per il dispositivo è necessario un indirizzo secondario. Se il dispositivo è sul bus seriale, SA deve essere modificato con OR 96 (60H) prima di usarlo; deve essere posto in A.

SETLFS: prepara i parametri per un file logico.
Chiamata: $FFBA (65466)
Comunic.: A, X e Y
Byte stack: 2
Prepara il LFN, l'indirizzo della periferica e SA per le altre routine KERNAL. LFN è usato come chiave di ricerca nella tabella dei file creata dalla OPEN. Gli indirizzi delle periferiche vanno da 0 a 30 e gli indirizzi superiori a 4 si riferiscono automaticamente al bus seriale. Gli indirizzi usati sono i seguenti: 0 per tastiera, 1 per cassetta, 2 per RS-232C, 3 per video, 4 per bus seriale (stampante), 8 per bus seriale (disco). Prima della chiamata si devono caricare i registri così: LFN in A, numero device in X, SA in Y (se SA non serve Y deve contenere 255).

SETMSG: controlla la stampa dei messaggi di controllo e di errore.
Chiamata: $FF90 (65424)
Comunic.: A
Byte stack: 2
Reg. usati: A
Il messaggio stampato dipende dal numero che si trova in A. Se il bit 6 è 1 i messaggi sono di controllo, se il bit 7 è 1 i messaggi sono di errore.

SETNAM: prepara il nome del file.
Chiamata: $FFBD (65469)
Comunic.: A, X e Y
Questa routine prepara il nome del file da utilizzare in una successiva OPEN, SAVE o LOAD.
Si deve porre in A la lunghezza del nome del file; in X e Y l'indirizzo del nome del file (byte basso prima del byte alto). Se non è richiesto il nome, si deve mettere 0 in A.

SETTIM: serve per inizializzare l'orologio interno.
Chiamata: $FFDB (65499)
Comunic.: A,X,Y
Byte stack: 2
L'orologio del sistema viene gestito dalla routine di interrupt che va in onda 60 volte al secondo. È costituito da 3 byte consecutivi e questo gli permette di contare fino a 5.184.000 sessantesimi di secondo, cioè 24 ore; a quel punto riparte da 0. Con questa routine si può caricare il numero desiderato nell'orologio, ponendolo nei 3 registri A, X e Y, in ordine dal byte più significativo al meno significativo.

SETTMO: attiva il flag di timeout sul bus IEEE
Chiamata: $FFA2 (65442)
Comunic.: A
Byte stack: 2
Se il flag di timeout è attivato, il Commodore 64 attende risposta da un dispositivo collegato alla porta IEEE per 64 millisecondi. Se entro questo tempo il dispositivo non risponde con un segnale di indirizzo dati valido (Data Address Valid = DAV) la comunicazione viene interrotta e si ha una segnalazione di errore. Chiamando questa routine il bit 7 dell'accumulatore indica se abilitare (0) o disabilitare (1) il flag di timeout.
Questa routine è utile solo se si possiede la scheda IEEE.

STOP: controlla se è premuto il tasto STOP.
Chiamata: $FFE1 (65505)
Comunic.: A
Reg. usati: A,X
Dopo la chiamata, se è stato trovato premuto lo STOP, si ha il flag Z in on. Se non e stato premuto i flag non vengono modificati e in A si trova un byte che rappresenta l'ultima riga della tastiera che è stata analizzata.

TALK: invia un comando di TALK attraverso il bus seriale.
Chiamata: $FFB4 (65460)
Comunic.: A
Errore: READST
Reg. usati: A
Prima della chiamata si deve porre in A il numero del dispositivo (da 4 a 30). Questo numero viene convertito nell'indirizzo del dispositivo e inviato al bus seriale.

TKSA: manda SA a un dispositivo che ha ricevuto il comando TALK.
Chiamata: $FF96 (65430)
Comunic.: A
Preparazione: TALK
Errore: READST
Reg. usati: A
Prima della chiamata SA deve essere posto in A. Può essere usata solo dopo TALK.

UDTIM: aggiorna l'orologio interno.
Chiamata: $FFEA (65514)
Byte stack: 2
Reg. usati: A, X
È chiamata dalla routine di interrupt di sistema 60 volte al secondo. Se l'utente gestisce le interruzioni in modo diverso, deve chiamarla ugualmente per aggiornare l'orologio, come anche deve chiamare la routine di STOP per far sì che il tasto corrispondente resti funzionante.

UNLSN: invia un comando di UNLISTEN a tutti i dispositivi del bus seriale.
Chiamata: $FFAE (65454)
Errore: READST
Reg. usati: A
Questo comando riguarda solo i dispositivi che si trovano in stato di LISTEN.

UNTLK: invia un comando di UNTALK a tutti i dispositivi del bus seriale.
Chiamata: $FFAB (65451)
Errore: READST
Reg. usati: A
Questo comando riguarda solo i dispositivi che sono in stato TALK.

VECTOR: aggiorna tutti i vettori del sistema che stanno in RAM.
Chiamata: $FF8D (65421)
Comunic.: X, Y
Byte stack: 2
Reg. usati: A, X e Y
Il funzionamento della routine dipende dal bit Carry. Se Carry=1 la routine usa l'indirizzo contenuto in X e Y come puntatore a una lista in RAM dove memorizza tutti i vettori del sistema. Se invece Carry=0 la lista puntata da X e Y serve per andare a scrivere nei vettori in RAM modificandone il contenuto. Conviene chiamare una prima volta la routine per leggere in area utente i contenuti dei registri, poi modificare quelli che servono e chiamare una seconda volta la routine per andare a riscrivere i vettori.

CODICI DI ERRORE

Le routine del KERNAL possono indicare errori sia utilizzando la routine READST sia tramite un valore restituito nell'accumulatore, con il bit di CARRY posto a 1. Quella che segue è una tabella dei possibili codici di errore con il relativo significato.

Tabella 1.8

COD	SIGNIFICATO
0	Routine interrotta dalla pressione di STOP
1	Troppi file aperti
2	File già aperto
3	File non aperto
4	File non trovato
5	Dispositivo non presente o non acceso
6	Il file indicato non è un file di input
7	Il file indicato non è un file di output
8	Manca il nome del file
9	Numero di dispositivo non valido
240	Variazione nell'indirizzo di fine memoria Allocazione o deallocazione del buffer RS-232

1.8.2 Il BASIC

In questo paragrafo riportiamo alcune tabelle con gli indirizzi esadecimali delle routine che eseguono le funzioni e i comandi BASIC, delle tavole di puntatori usati dall'Interprete, dei testi dei messaggi di errore e di servizio (anche del KERNAL), delle costanti usate per i calcoli e di parte delle routine utilizzate dall'Interprete BASIC per leggere dal testo del programma i parametri e convertirli nel formato adatto alle proprie necessità.

FUNZIONI E COMANDI	
ABS	BC58
AND	AFE9
ASC	B78B
ATN	E30E
CHR$	B6EC
CLOSE	E1C7
CLR	A65E
CMD	AA86
CONT	A857
COS	E264
DATA	A8F8
DEF	B3B3
DIM	B07E
END	A831
EXP	BFED
EXP (seguito)	E000
FN (sintassi)	B3E1
FN (valutazione)	B3F4
FOR	A742
FRE	B37D
GET	AB7B
GOSUB	A883
GOTO	A8A0
IF	A928
INPUT	ABBF
INPUT#	ABA5
INT	BCCC
LEFT$	B700
LEN	B77C

LET	A9A5
LET (intero)	A9C4
LET (reale)	A9D6
LET (stringa)	A9D9
LET (TI$)	A9E0
LIST	A69C
LOAD	E168
LOG	B9EA
MID$	B737
NEW	A642
NEXT	AD1E
NOT	AED4
ON	A94B
OPEN	E1BE
OR	AFE6
PEEK	B80D
POKE	B824
POS	B39E
PRINT	AAA0
PRINT (carattere)	AB3B
PRINT (stringa)	AB21
PRINT#	AA80
READ	AC06
REM	A93B
RESTORE	A81D
RETURN	A8D2
RIGHT$	B72C
RND	E097
RUN	A871
SAVE	A156
SGN	BC39
SIN	E26B
SQR	BF71
STOP	A82F
STR$	B465
SYS	E12A
TAN	E2B4
THEN (IF...THEN)	A940
VAL	B7AD
VERIFY	E165
WAIT	B82D

VETTORI, TABELLE E MESSAGGI

IND. DI RESET	A000-A001
IND. WARM START	A002-A003
"A000 CBM BASIC"	A004-A00B
IND. COMANDI BASIC	A00C-A050
IND. FUNZIONI	A052-A07E
PRIORITA' E INDIR. OPERATORI BINARI	A080-A09D
TAB. PAROLE CHIAVE	A09E-A128
TAB. FUNZIONI	A129-A19A
ALTRI COMANDI	A19B-A19D
MESSAGGI DI ERRORE	A19E-A327
IND. MESSAGGI ERR.	A328-A362
ALTRI MESSAGGI	A364-A389
MESSAGGI PER READ	ACFC-AD1D
INTESTAZIONE VIDEO	A45F-A4AC
TAB. BAUD-RATE PER MACCHINE INTERNAZ.	E4EC-E4FF
CODICI COLORE	E8DA-E8E9
IND. TAB. TASTIERA	EB79-EB80
TASTIERA NORMALE	EB81-EBC1
+SHIFT	EBC2-EC02
+CBM	EC03-EC43
+CONTROL	EC78-ECB8
VALORI REG. VIC-II	ECB9-ECE6
TESTO PER SHIFT+RUN/STOP	ECE7-ECEF
TAB. BYTE BASSI INDIRIZZI LINEE DELLO SCHERMO	ECF0-ED08
TESTO ERR. KERNAL	F0BD-F12A
VETTORI S.O.	FD30-FD4F
TAB. BAUD-RATE PER MACCHINE USA	FEC2-FED5
VETTORI 6510	FFFA-FFFF

COSTANTI NUMERICHE

PI	AEA8	
CONV. FLP-INTERO	B1A5	
1	B9BC	(FLP=FLoating-Point=Virgola mobile)
3 (1 byte)	B9C1	Se non diversamente specificato
0.434255942	B9C2	le costanti occupano 5 byte
0.576584541	B9C7	
0.961800759	B9CC	
2.88539007	B9D1	
0.5 * SQR(2)	B9D6	
SQR(2)	B9DB	
—0.5	B9E0	
LOG(2)	B9E5	
DIVISIONE PER 10	BAF9	
999999.9	BDB3	
999999999	BDB8	
1000000000	BDBD	
0.5	BF11	
—100000000	BF16	
+10000000	BF1A	
—1000000	BF1E	
+100000	BF22	
—10000	BF26	
+1000	BF2A	
—100	BF2E	
+10	BF32	
—1	BF36	
—10*6*10*6*10*60	BF3A	
+6*10*6*10*60	BF3E	
—10*6*10*60	BF42	
+6*10*60	BF46	
—10*60	BF4A	
60	BF4E	
1/LOG(2)	BFBF	
7 (1 byte)	BFC4	
Tabella polinomiale per EXP		BFC5-BFEC
NUMERI FLP PER RND		E08D-E096
NUMERI FLP PER SIN, COS E TAN		E2E0-E2EE
5 (1 byte)		E2EF
Tabella polinomiale per SIN		E2F0-E30D
11 (1 byte)		E33E
Tabella polinomiale per ATN		E33F-E37A
SEME INIZIALE RND		E3BA

ROUTINE	
Cerca blocco FOR nello stack	A38A
Cerca spazio e muove bytes	A3B8
Test 2*A byte liberi nello stack	A3FB
Errore OUT OF MEMORY	A435
Routine messaggi di errore BASIC in X il codice di errore	A437
Routine warm-start	A483
Accetta linea BASIC numerata	A49C
Cancella linea BASIC (ind.in $5F)	A4A9
Inserisce linea BASIC (ind.in $5F)	A4ED
Link linee BASIC	A533
Legge una linea nel buffer BASIC	A560
Spezza/riconosce TOKENS	A579
Cerca una linea BASIC —$14=numero di linea —CARRY=1/0 ——> trovata/non trovata —$5F=indirizzo della linea o della prima che segue	A613
Reset stack e puntatori	A67A
Puntatore car. a inizio programma	A68E
Stampa TOKENS	A717
Stampa KEYWORD	A737
Esegue prossima istruzione	A7AE
Esegue un'istruzione	A7E4
Esegue comando (codice in A)	A7ED
Cerca fine istruzione (:)	A906
Cerca fine linea ($00)	A909
Legge un numero decimale dal testo BASIC in $14	A96B
Stampa la stringa puntata da (A/Y) fino al primo byte $00	AB1E
Valuta un'espressione numerica nel testo BASIC	AD8A
Controlla se l'ultima espressione valutata era una stringa. Se no genera un errore: TYPE MISMATCH	AD8F
Valuta una qualunque espressione nel testo BASIC. Se il risultato è FLP va in FAC1. Se è intero va in FAC+3. Se è una stringa il puntatore alla descrizione della stringa va in FAC+3	AD9E
Cerca e valuta termine numerico	AE83
Valuta un'espressione tra parentesi	AEF1
Controlla se il prossimo carattere nel testo BASIC (puntato da $7A) è	
una parentesi chiusa ')'	AEF7
una parentesi aperta '('	AEFA
una virgola ','	AEFD
il carattere in A	AEFF

in caso contrario SYNTAX ERROR!		
Cerca il valore di una variabile il cui nome è nel testo BASIC		AF28
Converte TI in una stringa		AF48
Valuta una funzione		AFA7
Esegue un confronto		B016
confronto tra numeri		B01B
confronto tra stringhe		B02E
Legge il nome di una variabile		B08B
....e ne determina il puntatore		B0E7
Controlla il carattere in A CARRY=1 alfabetico		B113
Crea una nuova variabile		B11D
Converte (FAC) in intero in (A/Y)		B1AA
Valuta un'espressione nel testo come intero tra —32768 e 32767		B1B2
Converte (FAC) in intero in (FAC) tra 0 e 32767		B1BF
Cerca i parametri di una matrice		B1D1
Cerca una matrice in memoria		B218
Crea una matrice		B261
Calcola l'indirizzo dell'elemento di una matrice		B2EA
Converte da intero (A/Y) tra 0 e 32767 in FLP		B391
Converte da byte (Y) a FLP		B3A2
Esamina la stringa che inizia in (A/Y) e la copia in memoria		B487
Garbage collection delle stringhe		B526
Operatore '+' per le stringhe		B63D
Legge un numero tra 0 e 255 dal testo BASIC in X		B79B
Converte una stringa puntata da $22, lunghezza in A in FLP		B7B5
Legge due parametri dal testo, un indirizzo ($14) e un byte (X)		B7EB
Converte (FAC) in intero in $14 nell'intervallo 0-65535		B7F7
Somma 0.5 a FAC		B849
Sottrazione FLP	(FAC)=(A/Y)—(FAC)	B850
Ingresso sottrazione		B853
Somma FLP	(FAC)=(A/Y)+(FAC)	B867
Ingresso somma		B86A
Prodotto FLP	(FAC)=(FAC)*(A/Y)	BA28
Ingresso prodotto		BA2B
Per un byte	$26—$2A = (FAC)*A	BA59
Moltiplica FAC per 10		BAE2
Divide FAC per 10		BAFE
Divisione FLP	(FAC)=(A/Y)/(FAC)	BB0F
Ingresso divisione		BB12
Valore puntato da (A/Y) in FAC		BBA2
Copia FAC in $5C—$60		BBC7
Copia FAC in $57—$5B		BBC9
Copia FAC nella memoria ($49)		BBD0

Scrive FAC nella memoria (X/Y)	BBD4
Copia FAC2 in FAC	BBFC
Copia FAC arrotondato in FAC2	BC0C
Arrotonda FAC	BC1B
Scrive in A il segno di FAC	BC2B
Copia A (intero con segno) in FAC)	BC3B
Converte (FAC) in intero a 4 byte in FAC+1	BC9B
Azzera FAC	BCE9
Converte una stringa ASCII, dal testo BASIC in FAC	BCF3
Stampa IN e numero di linea	BDC2
Stampa il numero di linea (A/X)	BDCD
Converte FAC in stringa nello STACK	BDDD
Trattamento errori di I/O BASIC	E0F9
Seleziona parametri per LOAD/VERIFY/SAVE	E1D4
Seleziona parametri per OPEN/CLOSE	E219
Trattamento errori BASIC/KERNAL	E83B
Routine di RESET	E394
Inizializza l'interprete BASIC	E3BF
Stampa messaggio iniziale BASIC ed effettua NEW	E422

1.9 PROGRAMMI ESEMPIO

Quelli che seguono sono alcuni programmi per esemplificare quanto visto fin qui. Tutte le routine in linguaggio macchina presentate, normalmente molto brevi, sono listate nel formato usato dal comando 'D' del monitor ed accompagnate da un caricatore scritto in BASIC che contiene anche un piccolo esempio di uso.
Nelle routine che seguono:

- la prima colonna riporta un numero in esadecimale, preceduto da virgola, che è l'indirizzo del primo byte dell'istruzione scritta nella riga;

● le successive tre colonne (che possono ridursi solo a una o due) rappresentano in codice macchina, byte a byte, l'istruzione in esadecimale;

● le ulteriori colonne rappresentano l'istruzione in linguaggio simbolico assembler (dove $ rappresenta un numero esadecimale);

● i commenti sono a parte nell'ultima colonna.

1.9.1 RAMPEEK

Il primo esempio che ti presentiamo è l'uso di USR per poter esaminare le RAM presenti sotto le due ROM di BASIC e Sistema Operativo. Il principio di funzionamento è abbastanza semplice: si converte in intero l'argomento passato come parametro, si disabilitano le due ROM e si legge un byte dalla memoria. Quindi si riabilitano le ROM e si mette il valore del byte letto nel FAC per poter essere restituito come valore della funzione. Come puoi notare, il codice è piuttosto semplice, poiché utilizziamo le routine del BASIC.

RAMPEEK assembler (la routine è rilocabile)

```
,0340   20  f7 b7   jsr  $b7f7    converte FAC in intero
,0343   a0  00      ldy  #$00     prepara Y per LDA(),Y
,0345   a5  64      lda  $64        ed inverte i byte del risultato
,0347   a6  65      ldx  $65        per ottenere l'indirizzo nel formato 6510
,0349   85  65      sta  $65
,034b   86  64      stx  $64
,034d   78          sei           disabilita interruzioni
,034e   a5  01      lda  $01       ...e ROM
,0350   29  fc      and  #$fc
,0352   85  01      sta  $01
,0354   b1  64      lda  ($64),y  legge un byte
,0356   a8          tay
,0357   a5  01      lda  $01      riabilita ROM
,0359   09  03      ora  #$03
,035b   85  01      sta  $01
,035d   58          cli           ...e interruzioni
,035e   a9  00      lda  #$00     converte l'intero in A/Y
,0360   4c  91 b3   jmp  $b391      nel FAC e torna al BASIC
```

Il programma BASIC che segue carica in memoria la routine e, dopo aver scritto dei dati sotto la ROM del BASIC prova a rileggerli utilizzando PEEK e USR. Naturalmente solo con la funzione appena definita si riescono ad ottenere i dati nella RAM.

```
5  REM *** A1.91
10 GOSUB 10000
20 POKE 785,64: POKE 786,3
30 FOR K=0 TO 10:POKE 40960+K,K: NEXT
40 PRINT CHR$(147)"PEEK","USR"
50 FOR K=0 TO 10
60 PRINTPEEK(40960+K),USR(40960+K)
70 NEXT
80 END
10000 REM *** CARICATORE
10010 K=0: I=832
10020 READ A$: IF A$="*" THEN RETURN
10030 POKE I+K,VAL(A$):K=K+1:GOTO10020
10040 DATA32,247,183,160,0,165,100,166
10050 DATA101,133,101,134,100,120,165,1
10060 DATA41,252,133,1,177,100,168,165
10070 DATA1,9,3,133,1,88,169,0
10080 DATA76,145,179,*
```

1.9.2 SCROLLING a destra

In questo secondo esempio manipoliamo un po' lo schermo, preparando una routine che effettua lo scrolling dello schermo di un carattere verso destra. Lo schema di funzionamento della routine è il seguente:

(1) inizializza i puntatori

(2) effettua lo scrolling a destra su una linea

(3) se era l'ultima linea esce

(4) altrimenti torna al punto 2.

SCROLL A DESTRA assembler (la routine è rilocabile)

```
,9ce5  78        sei          disabilita interruzioni
,9ce6  a9  00    lda #$00
,9ce8  85  fb    sta $fb      i due puntatori puntano al primo byte ...
,9cea  85  fb    sta $fd
,9cec  a9  04    lda #$04     del video
,9cee  85  fc    sta $fc
,9cf0  a9  d8    lda #$d8     e della mappa colore
,9cf2  85  fe    sta $fe
,9cf4  a2  19    ldx #$19     X=numero di righe: 25
,9cf6  a0  26    ldy #$26     Y=numero colonne: 38
,9cf8  b1  fb    lda ($fb),y  copia un carattere nella posizione che segue
,9cfa  c8        iny
,9cfb  91  fb    sta ($fb),y
,9cfd  88        dey
,9cfe  b1  fd    lda ($fd),y  e fa lo stesso per il colore del carattere
,9d00  c8        iny
,9d01  91  fd    sta ($fd),y
,9d03  88        dey
,9d04  88        dey          decrementa Y e se non è negativo continua
,9d05  10  f1    bpl $9cf8
,9d07  a9  20    lda #$20     mette uno spazio nella prima posizione
,9d09  c8        iny            della linea
,9d0a  91  fb    sta ($fb),y
,9d0c  ca        dex          decrementa X e termina se era l'ultima linea
,9d0d  f0  11    beq $9d20
,9d0f  18        clc          altrimenti aggiunge 40 ad entrambi
,9d10  a9  28    lda #$28       i puntatori
,9d12  65  fb    adc $fb
,9d14  85  fd    sta $fd
,9d16  85  fb    sta $fb
,9d18  90  dc    bcc $9cf6
,9d1a  e6  fe    inc $fe
,9d1c  e6  fc    inc $fc
,9d1e  b0  d6    bcs $9cf6    e torna allo scrolling
,9d20  58        cli
,9d21  60        rts          fine
```

Il programma BASIC che segue è solo il caricatore. A differenza del precedente, però, queso richiede che i dati siano esadecimali, a parte quello che indica l'indirizzo di caricamento, un numero decimale preceduto dal simbolo "!".
Nella linea 10, inoltre, viene modificato il puntatore di fine memoria per BASIC e S.O. in modo da riservare spazio in fondo alla RAM per la nostra routine.

Dopo l'esecuzione del caricatore, prova a dare, in modo immediato, il comando:

FOR K=0 to 39: SYS 40165: NEXT

```
1   REM *** A1.92
10  POKE51,228:POKE52,156:POKE55,228:POKE56,156
20  AD=40000:K=0:RESTORE
30  K=K+1:READA$:IFA$="*"THEN100
40  IFASC(A$)<>ASC("!")THEN50
45  AD=VAL(MID$(A$,2)):ON(2+(AD=0))GOTO95,30
50  A=ASC(LEFT$(A$,1))
55  IFA<48ORA>57ANDA<65ORA>70THEN95
60  A=A-55-7*(A<58)
70  B=ASC(RIGHT$(A$,1))
75  IFB<48ORA>57ANDA<65ORA>70THEN95
80  B=B-55-7*(B<58)
90  POKE AD,A*16+B:AD=AD+1:GOTO30
95  PRINT "DATO #"K"■ ERRATO!":STOP
100 END
2000 DATA !40165
2010 DATA 78,A9,00,85,FB,85,FD,A9,04
2020 DATA 85,FC,A9,D8,85,FE,A2,19,A0,26
2030 DATA B1,FB,C8,91,FB,88,B1,FD,C8
2040 DATA 91,FD,88,88,10,F1
2050 DATA A9,20,C8,91,FB
2060 DATA CA,F0,11,18,A9,28,65,FB,85,FD
2070 DATA 85,FB,90,DC,E6,FE,E6,FC,B0,D6
2080 DATA 58,60,*
```

1.9.3 RESTORE num-linea

Molte routine in linguaggio macchina sono scritte per rimediare a carenze dovute a errori, dimenticanze o tentativi di mantenere la compatibilità tra i diversi computer effettuati dagli autori del software di base.
Un esempio è l'istruzione BASIC "RESTORE" che riporta il puntatore ai DATA all'inizio del programma, quando ormai quasi tutti i BASIC consentono di indicare a quale linea riposizionare il puntatore.
Così, sfruttando le routine già presenti in memoria, abbiamo scritto una routine che simula l'istruzione:

RESTORE num-linea

e che va utilizzata con la sintassi:

SYS IND, num-linea

dove IND, nel nostro caso 832 ($0340) ma non fa differenza poiché la routine è rilocabile, è l'indirizzo in cui la routine è stata caricata.

Attenzione, poiché questa routine lavora direttamente sul testo BASIC non funzionerà più nel caso compili il tuo programma.

RESTORE assembler (la routine è rilocabile)

```
,0340   20  fd  ae  jsr  $aefd   salta la virgola dopo IND
,0343   20  6b  a9  jsr  $a96b   legge il numero di linea
,0346   20  13  a6  jsr  $a613   e cerca la linea dei DATA
,0349   b0  05      bcs  $0350   se non la trova stampa il messaggio
,034b   a2  11      ldx  #$11      ?UNDEF 'D STATEMENT
,034d   4c  37  a4  jmp  $a437
,0350   a5  5f      lda  $5f     altrimenti decrementa di uno l'indirizzo tro-
,0352   e9  01      sbc  #$01      vato (è necessario perché la prossima
,0354   85  41      sta  $41       READ funzioni!) e lo trascrive nel
,0356   a5  60      lda  $60       puntatore in pagina 0
,0358   e9  00      sbc  #$00
,035a   85  42      sta  $42
,035c   a5  14      lda  $14     Poi aggiorna anche il numero di linea
,035e   85  3f      sta  $3f       della istruzione DATA corrente
,0360   a5  15      lda  $15
,0362   85  40      sta  $40
,0364   60          rts
```

Il caricatore BASIC che segue deve leggere a vuoto i 20 dati usati nella dimostrazione prima di poter trovare i codici della routine. Ma dopo aver installato tali codici, è in grado di leggere le informazioni nelle linee DATA persino andando all'indietro.

```
1  REM *** A1.93
10 RESTORE
15 REM SALTA I DATI DELL'ESEMPIO
20 FOR K=1 TO 20:READ A$:NEXT
25 GOSUB 10000
30 PRINTCHR$(147);
35 SYS 832,400
40 FORK=1TO5:READA$:PRINTA$:NEXT
45 SYS 832,300
50 FORK=1TO5:READA$:PRINTA$:NEXT
55 SYS 832,200
60 FORK=1TO5:READA$:PRINTA$:NEXT
65 SYS 832,100
70 FORK=1TO5:READA$:PRINTA$:NEXT
75 END
100 DATA 1,2,3,4,5
200 DATA A1,A2,A3,A4,A5
300 DATA B1,B2,B3,B4,B5
400 DATA C1,C2,C3,C4,C5
10000 REM *** CARICATORE
10010 K=0: I=832
10020 READ A$: IF A$="*" THEN RETURN
10030 POKE I+K,VAL(A$):K=K+1:GOTO10020
10040 DATA32,253,174,32,107,169,32,19
10050 DATA166,176,5,162,17,76,55,164
10060 DATA165,95,233,1,133,65,165,96
10070 DATA233,0,133,66,165,20,133,63
10080 DATA165,21,133,64,96,*
```

1.9.4 AT

Nel primo esempio di uso del comando SYS abbiamo richiamato la routine KERNAL PLOT per posizionare il cursore ovunque sullo schermo. Abbiamo però dovuto prima memorizzare nei byte di salvataggio dei registri i valori adeguati. È certamente più comodo poter utilizzare una istruzione del tipo:

SYS IND,riga,colonna

e ottenere lo stesso risultato.
La routine effettua il controllo sulla validità dei parametri passati dopo di che chiama la routine PLOT, senza però passare dalla tabella KERNAL.

AT assembler (la routine è rilocabile)

```
,0340  20 fd ae  jsr  $aefd   salta la virgola
,0343  20 9e b7  jsr  $b79e   legge dal testo BASIC un numero tra 0
                                e 255 in X
,0346  e0 19     cpx  #$19    e se è maggiore di 24 stampa il messaggio
,0348  30 03     bmi  $034d     ?ILLEGAL QUANTITY ERROR
,034a  4c 48 b2  jmp  $b248     (salta a una routine del BASIC
                                che richiama quella di errore)
,034d  8a        txa
,034e  48        pha          se invece è valido lo salva nello STACK
,034f  20 fd ae  jsr  $aefd
,0352  20 9e b7  jsr  $b79e   legge dal testo BASIC il numero di colonne
,0355  e0 28     cpx  #$28    e controlla se è accettabile (<40)
,0357  30 03     bmi  $035c
,0359  4c 48 b2  jmp  $b248
,035c  8a        txa          trascrive la colonna in Y
,035d  a8        tay
,035e  68        pla          recupera la riga in X
,035f  aa        tax
,0360  18        clc          e chiama la routine di PLOT
,0361  4c 0a e5  jmp  $e50a
```

Dopo aver memorizzato il codice nel buffer del nastro, il caricatore riempie di "*" lo schermo, dal basso verso l'alto, evitando le prime e ultime righe e colonne.

```
10 REM *** A1.94
20 GOSUB 10000
30 FOR R= 23 TO 1 STEP -1
40 FOR C= 38 TO 1 STEP -1
50 SYS 832,R,C: PRINT"*";
60 NEXT C,R
70 END
10000 REM *** CARICATORE
10010 K=0: I=832
10020 READ A$: IF A$="*" THEN RETURN
10030 POKE I+K,VAL(A$):K=K+1:GOTO10020
10040 DATA32,253,174,32,158,183,224,25
10050 DATA48,3,76,72,178,138,72,32
10060 DATA253,174,32,158,183,224,40,48
10070 DATA3,76,72,178,138,168,104,170
10080 DATA24,76,10,229,*
```

1.9.5 BSAVE

Il BASIC del COMMODORE 64 non dispone, tra le altre cose, delle istruzioni BLOAD e BSAVE dove la 'B' sta per 'Binario' cioè ad immagine della memoria. Nella versione del 128 (e nel SIMON'S BASIC, II versione) tali comandi sono stati aggiunti. Con un po' di lavoro, comunque è possibile supplire a queste carenze. Ti proponiamo una routine che legge e controlla i parametri necessari e quindi chiama la routine di salvataggio della memoria.
La sintassi è:

SYS IND,nome-file,DN,SA,ind.INIZIO,ind.FINE

e nessun parametro è opzionale.
Sulla stessa falsariga, se ti interessa, puoi scrivere una routine che esegua il caricamento in memoria di file in formato binario.

BSAVE assembler (la routine è rilocabile)

```
,02a7  20 fd ae  jsr $aefd   salta la virgola
,02aa  20 d4 e1  jsr $e1d4   e legge i parametri per
                               SAVE/VERIFY/LOAD
,02ad  20 fd ae  jsr $aefd   salta la virgola
,02b0  20 8a ad  jsr $ad8a   e valuta un'espressione
,02b3  20 f7 b7  jsr $b7f7   che deve corrispondere a un numero tra 0 e
                               65535, l'indirizzo di INIZIO,

,02b6  84 fb     sty $fb     che memorizza in $fb-$fc
,02b8  85 fc     sta $fc
,02ba  20 fd ae  jsr $aefd   salta la virgola
,02bd  20 8a ad  jsr $ad8a   e cerca l'indirizzo
,02c0  20 f7 b7  jsr $b7f7   FINE (che rimane in A/Y)
,02c3  c5 fc     cmp$fc      controlla che l'indirizzo INIZIO
,02c5  f0 04     beq $02cb     non superi FINE
,02c7  10 09     bpl $02d2
,02c9  30 04     bmi $02cf
,02cb  c4 fb     cpy $fb     (prima il byte alto e poi se serve
,02cd  10 03     bpl $02d2     il byte basso)
,02cf  4c 48 b2  jmp $b248   nel qual caso stampa
                               ?ILLEGAL QUANTITY ERROR
,02d2  48        pha           scambia i valori tra i registri in modo
,02d3  98        tya           da avere l'indirizzo FINE in (Y/X)
,02d4  aa        tax
,02d5  68        pla
,02d6  a8        tay
,02d7  e8        inx         aggiunge 1 all'indirizzo FINE
,02d8  d0 01     bne $02db     (come vuole SAVE)
,02da  c8        iny
,02db  a9 fb     lda #$fb    A punta a INIZIO
,02dd  20 d8 ff  jsr $ffd8   chiama la routine SAVE via KERNAL
,02e0  90 03     bcc $02e5   se ci sono errori li segnala
,02e2  4c f9 e0  jmp $e0f9
,02e5  60        rts         altrimenti termina
```

Il caricatore BASIC non contiene anche una dimostrazione.
Ti consigliamo, dopo averlo eseguito, di provare a dare il comando:

SYS 679, "prova2",8,1,1024,2023

e poi provare a rileggere con:

LOAD"prova2",8,1

```
10 REM A1.95
20 REM ROUTINE SALVATAGGIO MEMORIA
30 REM
40 GOSUB 10000
50 PRINTCHR$(147)"SINTASSI:"
60 PRINT"SYS 679,NOMEFILE,DN,SA,INIZIO,FINE"
70 END
10000 REM *** CARICATORE
10010 K=0: I=679
10020 READ A$: IF A$="*" THEN RETURN
10030 POKE I+K,VAL(A$):K=K+1:GOTO10020
10040 DATA32,253,174,32,212,225,32,253
10050 DATA174,32,138,173,32,247,183,132
10060 DATA251,133,252,32,253,174,32,138
10070 DATA173,32,247,183,197,252,240,4
10080 DATA16,9,48,4,196,251,16,3
10090 DATA76,72,178,72,152,170,104,168
10100 DATA232,208,1,200,169,251,32,216
10110 DATA255,144,3,76,249,224,96,*
```

1.9.6 Modifica INTERRUPT: FLASHING

Come ultimo esempio, una modifica della routine di gestione delle interruzioni, composta da due parti:

- la prima modifica il vettore in RAM in modo che punti alla nuova routine di gestione interruzioni;
- la seconda è la vera e propria routine di gestione.

Le operazioni svolte dalla seconda parte sono, in questo esempio, molto semplici: la routine si propone di rendere possibile il LAMPEGGIAMENTO (FLASHING) dei caratteri dello schermo, non ottenibile con dei comandi al VIC II.
Per fare ciò controlla il contenuto del byte all'indirizzo 2:

- se è dispari (primo bit a 0) il FLASHING è disabilitato, quindi salta alla normale routine di gestione delle interruzioni;
- altrimenti decrementa un contatore;

- se questo non è tornato al valore 0 passa alla normale gestione delle interruzioni;
- quando il contatore diventa 0 viene ri-inizializzato e la mappa colore viene scandita: il codice di ogni carattere il cui colore è quello prestabilito viene scambiato con quello del suo inverso.
- dopo di ciò si richiama comunque la normale gestione delle interruzioni.

FLASHING assembler (la routine NON È rilocabile. Vanno cambiate le istruzioni indicate con '*')

```
,9c40  78                 sei          installa la routine
,9c41  a9  4d            *lda #$4d     mette nel vettore CINV l'indirizzo $9C4D
,9c43  8d  14  03         sta $0314
,9c46  a9  9c            *lda #$9c
,9c48  8d  15  03         sta $0315
,9c4b  58                 cli
,9c4c  60                 rts
,9c4d  a5  02             lda $02      legge il byte in $02
,9c4f  29  01            and #$01      se il primo bit è 0 salta alla fine
,9c51  f0  49            beq $9c9c
,9c53  ce  a7  02        dec $02a7     decrementa il contatore
,9c56  d0  44            bne $9c9c     se non è 0 FINE
,9c58  ad  a8  02         lda $02a8    ri-inizializza il contatore
,9c5b  8d  a7  02         sta $02a7
,9c5e  a9  e7             lda #$e7     inizializza due puntatori alla fine
,9c60  85  fb             sta $fb        della mappa video e colore
,9c62  a9  db             lda #$db
,9c64  85  fc             sta $fc
,9c66  a9  e7             lda #$e7
,9c68  85  fd             sta $fd
,9c6a  a9  07             lda #$07
,9c6c  85  fe             sta $fe
,9c6e  a0  00             ldy #$00     prepara Y per LDA(),Y
,9c70  b1  fb             lda ($fb),y  legge un byte dalla mappa colore
,9c72  29  0f            and #$0f
,9c74  cd  a9  02        cmp $02a9     e se è quello da far lampeggiare
,9c77  d0  0c            bne $9c85
,9c79  b1  fd             lda ($fd),y  inverte il bit 7 del codice del carattere
,9c7b  10  04            bpl $9c81       (era più semplice usare semplicemente
,9c7d  29  7f            and #$7f        una EOR $80)
,9c7f  10  02            bpl $9c83
,9c81  09  80            ora #$80
```

```
,9c83   91  fd        sta ($fd),y  e lo riscrive sul video
,9c85   c6  fb        dec $fb      decrementa di 1 entrambi i puntatori
,9c87   c6  fd        dec $fd
,9c89   a5  fd        lda $fd
,9c8b   c9  ff        cmp #$ff
,9c8d   d0  e1        bne $9c70
,9c8f   a5  fe        lda $fe      se $fd-$fe puntava al primo byte
                                     dello schermo
,9c91   c9  04        cmp #$04     allora ha finito, altrimenti prosegue
,9c93   f0  07        beq $9c9c
,9c95   c6  fe        dec $fe
,9c97   c6  fc        dec $fc
,9c99   38            sec
,9c9a   b0  d4        bcs $9c70
,9c9c   4c  31  ea    jmp $ea31    FINE (salta alla normale routine di gestione
                                     delle interruzioni
```

Il caricatore BASIC contiene le informazioni necessarie per utilizzare la nuova routine (il colore scelto per lampeggiare è il nero).
Come puoi notare, alla linea 8 riserva quasi 1K di memoria RAM alla fine di quella normalmente disponibile per il BASIC, anche se la routine occupa molto meno spazio. Questo è stato fatto pensando di poter modificare e allungare la routine in un secondo tempo, aggiungendovi altre funzioni quali, ad esempio, il lampeggiamento di SFONDO e BORDO, il rilevamento dello stato dei JOYSTICK o lo sfruttamento dei tasti funzione.

```
0 REM A1.96  - FLASH
1 REM
2 REM POKE 2,0=OFF - POKE 2,1=ON
3 REM
4 REM ---- 679      - CONT. RITARDO
5 REM POKE 680,N    - OGNI N JIFFIES
6 REM POKE 681,C    - COLORE LAMPEGGIO
7 REM
8 POKE51,63:POKE52,156:POKE55,63:POKE56,156
10 AD=40000
20 READA$:IFA$="*"THEN100
30 IFASC(A$)=ASC("!")THENAD=VAL(MID$(A$,2)):GOTO20
40 A=ASC(LEFT$(A$,1)):A=A-55-7*(A<58)
50 B=ASC(RIGHT$(A$,1)):B=B-55-7*(B<58)
```

```
60 POKE AD,A*16+B:AD=AD+1:GOTO20
100 PRINTCHR$(147);
105 PRINT"SYS 40000 INSTALLA LA NUOVA ROUTINE"
110 PRINT
115 PRINT"RESTORE RIPRISTINA LA NORMALE ROUTINE"
120 PRINT
125 PRINT"POKE 2,1 ABILITA FLASHIMG"
130 PRINT
135 PRINT"POKE 2,0 DISABILITA FLASHING"
140 POKE2,0:POKE679,20:POKE680,20:POKE681,0
150 SYS 40000
155 PRINT:PRINT:PRINT:PRINT"ROUTINE INSTALLATA !"
160 END
1000 DATA !40000,78,A9,4D,8D,14,03
1010 DATA A9,9C,8D,15,03,58,60
1020 DATA !40013
1030 DATA A5,02,29,01,F0,49,CE,A7,02
1040 DATA D0,44,AD,A8,02,8D,A7,02
1050 DATA A9,E7,85,FB,A9,DB,85,FC
1060 DATA A9,E7,85,FD,A9,07,85,FE
1070 DATA A0,00,B1,FB,29,0F,CD,A9,02
1080 DATA D0,0C,B1,FD,10,04,29,7F
1090 DATA 10,02,09,80,91,FD,C6,FB
1100 DATA C6,FD,A5,FD,C9,FF,D0,E1
1110 DATA A5,FE,C9,04,F0,07,C6,FE
1120 DATA C6,FC,38,B0,D4,4C,31,EA,*
```

1.9.7 Un paio di consigli

Come avrai potuto notare ci sono delle aree di memoria "privilegiate" in cui questi programmi lavorano.
Una di queste è il buffer della cassetta: un'ottima zona di memoria da utilizzare, se si lavora prevalentemente su disco. C'è poi il gruppo di 88 byte che inizia all'indirizzo 679 che abbiamo usato per la routine di salvataggio memoria. In realtà questa zona di memoria viene sfruttata per lo più dalle diverse espansioni del BASIC per memorizzare le nuove variabili e i parametri di lavoro.
Naturalmente (e l'abbiamo fatto in un paio di esempi) è anche possibile riservarsi parte della memoria che precede la ROM dell'Interprete BASIC (40960=$A000).
Una zona che invece non abbiamo toccato è il blocco di 4K che inizia in 49152 ($C000), usatissimo invece dalle varie routine in circolazione (monitor, assemblatori, DOS-wedge, mini-espansioni, etc.) e che, se non devi utilizzare alcuna delle suddette routine, è a tua disposizione.
Ancora, SOTTO le ROM del BASIC e del KERNAL si nascondono ben 16K di RAM.

Per richiamare routine scritte sotto le ROM è sufficiente "rubare" una decina di byte in una qualche zona di memoria in modo da creare una piccola routine di lancio che disabiliti la ROM in questione, chiami la routine e, al termine, rimetta tutto a posto.
Altro uso per queste due zone di memoria è quello di MODIFICARE il BASIC ed il KERNAL copiandone prima i contenuti in RAM, alterandoli dove necessario, e quindi disabilitare le ROM. Ci si ritrova con un computer ALL-RAM.

Spesso quando si scrivono programmi in linguaggio macchina si devono affrontare problemi di ristrettezza di spazio in contrapposizione alla necessità di ottenere la massima velocità (applicazioni grafiche) o una perfetta temporizzazione (operazioni con il nastro).

Per risolvere il problema dello spazio è possibile sfruttare alcune caratteristiche delle istruzioni del 6510. Se provi ad esempio a disassemblare la routine che cerca i caratteri "(", ")", "," e (A) nel testo BASIC (vedi elenco routine) ottieni dei risultati come quelli che seguono, a prima vista senza senso:

```
AEF7  A9 29     LDA  #$29      $29 è il codice di ")"
AEF9  2C A9 28  BIT  28A9
AEFC  2C A9 2C  BIT  2CA9
AEFF  A0 00     LDY  #$00
AF01  D1 7A     CMP  ($7A),Y
..... etc......
```

Tutto, invece, è regolare, infatti l'istruzione BIT non fa altro che effettuare un test (magari inutile) tra l'accumulatore e la memoria modificando SOLO i FLAG. In questo pezzo di programma, che va quindi interpretato come segue, questa istruzione è utilizzata per IGNORARE altre due istruzioni 6510:

```
AEF7  A9 29     LDA  #$29      $29 è il codice di ")"
AEF9  2C                       codice BIT
AEFA  A9 28     LDA  #$28      $28 è il codice di "("
AEFC  2C                       codice BIT
AEFD  A9 2C     LDA  #$2C      $2C è il codice di "."
AEFF  A0 00     LDY  #$00
AF01  D1 7A     CMP  ($7A),Y
..... etc......
```

Un uso analogo si fa dell'altra forma di istruzione BIT, quella relativa a un indirizzo in pagina 0 (codice $24) utilizzata per ignorare istruzioni a un solo byte tipo CLC, INX, RTS, DEX etc.

Per risparmiare qualche altro byte si può fare qualche operazione che sprechi tempo (poco) e non crei effetti secondari:

```
C000   LDX   $02
C002   STX   $02
C004   RTS
```

questo frammento di codice legge in X il valore nel byte di indirizzo 2, se vi si arriva in $C000, modifica il byte all'indirizzo 2 se vi si arriva in $C002.

Per rendere più veloci i calcoli può essere utile, quando possibile, preparare delle tabelle che contengono valori pre-calcolati. Molto spesso l'uso di tabelle semplifica anche il codice da scrivere.

Per rendere il più facilmente rilocabile un programma (rilocabile significa che può essere trascritto, senza richiedere modifiche per funzionare, in qualunque zona della memoria) puoi sostituire qualche istruzione di salto, nell'intervallo di un centinaio di byte in avanti o indietro con una coppia di istruzioni del tipo:

```
.CLC                ; azzera il carry
.BCC     indirizzo  ; se il carry è a 0 salta
```

oppure:

```
.SEC                ; mette a 1 il carry
.BCS     indirizzo  ; se il carry è a 1 salta
```

o utilizzando un qualunque altro FLAG. Ricordati anche che una sequenza del tipo:

```
.LDA      #$00
.BEQ     indirizzo
```

provoca sempre una diramazione.

TABELLA 1

ISTRUZIONI ASSEMBLER IN ORDINE ALFABETICO

Nella tabella delle pagine seguenti sono riportate le istruzioni ASSEMBLER in ordine alfabetico.
Ogni riga riguarda un'istruzione, riportata in ordine alfabetico per codice mnemonico. Per ogni istruzione viene data in colonna 2 la descrizione sintetica dell'operazione. Seguono sulle colonne successive, per ogni tipo di indirizzamento, il codice esadecimale dell'istruzione, il numero dei cicli di clock impiegati per eseguirla e il numero dei byte occupati. Nelle ultime 6 colonne sono indicati i FLAG influenzati dall'esecuzione dell'istruzione.

ISTRUZIONI		immediato			assoluto			pag zero			accum			implicito			(ind,X)			(ind),Y			pag 0,X			assol,X			assol,Y			relativo			indiretto			pag 0,Y			FLAG
mnemo	operazione	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	N Z C I D V
ADC	A←A+M+C * ***	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										X X X X
AND	A←A∧M *	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										X X X
ASL	C←[Z 7 … 0]←0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													X X X
BCC	salta se C=0**																															90	2	2							
BCS	salta se C=1**																															B0	2	2							
BEQ	salta se Z=1**																															F0	2	2							
BIT	A∧M				2C	4	3	24	3	2																															M7X M6
BMI	salta se N=1**																															30	2	2							
BNE	salta se Z=0**																															D0	2	2							
BPL	salta se N=0**																															10	2	2							
BRK	vedi PAR. 1.4													00	7	1																									1
BVC	salta se V=0**																															50	2	2							
BVS	salta se V=1**																															70	2	2							
CLC	C←0													18	2	1																									0
CLD	D←0													D8	2	1																									0
CLI	I←0													58	2	1																									0
CLV	V←0													B8	2	1																									0
CMP	A-M *	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										X X X
CPX	X-M	E0	2	2	EC	4	3	E4	3	2																															X X X
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2																															X X X
DEC	M←M-1				CE	6	3	C6	5	2													D6	6	2	DE	7	3													X X
DEX	X←X-1													CA	2	1																									X X
DEY	Y←Y-1													88	2	1																									X X
EOR	A←A∨M *	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										X X
INC	M←M+1				EE	6	3	E6	5	2													F6	6	2	FE	7	3													X X
INX	X←X+1													E8	2	1																									X X
INY	Y←Y+1													C8	2	1																									X X
JMP	salta				4C	3	3																												6C	5	3				
JSR	salta a sotprg				20	6	3																																		
LDA	A←M *	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										X X

ISTRUZIONI		immediato			assoluto			pag zero			accum			implicito			(ind,X)			(ind),Y			pag 0,X			assol,X			assol,Y			relativo			indiretto			pag 0,Y			FLAG
mnemo	operazione	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	CO	N	#	N Z C I D V
LDX	X←M *	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2	X X
LDY	Y←M *	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3													X X
LSR	0→[7 ▯]→C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0 X X
NOP	nessuna													EA	2	1																									
ORA	A←AVM	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										X X
PHA	SK←A SP←SP-1													48	3	1																									
PHP	SK←P SP←SP-1													08	3	1																									
PLA	A←SK SP←SP+1													68	4	1																									X X
PLP	P←SK SP←SP+1													28	4	1																									X X X X X X
ROL	C←[7 ▯]←C				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													X X X
ROR	C→[7 ▯]→C				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													X X X
RTI	VEDI PAR. 1.4													40	6	1																									X X X X X X
RTS	VEDI PAR. 1.4													60	6	1																									
SBC	A←A-M-$\bar{C}$ *	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										X X X X
SEC	C←1													38	2	1																									1
SED	D←1													F8	2	1																									1
SEI	I←1													78	2	1																									1
STA	M←A				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3										
STX	M←X				8E	4	3	86	3	2																												96	4	2	
STY	M←Y				8C	4	3	84	3	2													94	4	2																
TAX	X←A													AA	2	1																									X X
TAY	Y←A													A8	2	1																									X X
TSX	X←S													BA	2	1																									X X
TXA	A←X													8A	2	1																									X X
TXS	S←X													9A	2	1																									
TYA	A←Y													98	2	1																									X X

* SE A CAVALLO DI PAGINA N=N+1
** SE VERIFICATO N=N+1
SE VERIFICATO E SALTA OLTRE LA PAGINA N=N+2
*** IN MODO DECIMALE IL FLAG DI ZERO NON E' VALIDO

X=REGISTRO X
Y=REGISTRO Y
A=ACCUMULATORE
M=OPERANDO
SK=CELLA DELLO STACK
SP=STACK POINTER

\+ SOMMA
\- SOTTRAZIONE
∧ AND LOGICO
∨ OR LOGICO
∨ OR ESCLUSIVO

X MODIFICA
M7 BIT 7 DELLA CELLA CONSIDERATA
M6 BIT 6 DELLA CELLA CONSIDERATA
N NUMERO DI CICLI DI CLOCK
\# NUMERO DI BYTES

TABELLA 2

ISTRUZIONI ASSEMBLER IN ORDINE CODICE OPERATIVO

Alla pagina che segue diamo l'elenco delle istruzioni ASSEMBLER in ordine di codice operativo.

Le abbreviazioni usate sono:

IND = indirizzo a 1 byte
PAG0 = indirizzo a 1 byte
ASS = indirizzo a 2 byte
IMM = dato a 1 byte
ACC = accumulatore

00	BRK		40	RTI		80	???		C0	CPY	IMM
01	ORA	(IND,X)	41	EOR	(IND,X)	81	STA	(IND,X)	C1	CMP	(IND,X)
02	???		42	???		82	???		C2	???	
03	???		43	???		83	???		C3	???	
04	???		44	???		84	STY	PAG0	C4	CPY	PAG0
05	ORA	PAG0	45	EOR	PAG0	85	STA	PAG0	C5	CMP	PAG0
06	ASL	PAG0	46	LSR	PAG0	86	STX	PAG0	C6	DEC	PAG0
07	???		47	???		87	???		C7	???	
08	PHP		48	PHA		88	DEY		C8	INY	
09	ORA	IMM	49	EOR	IMM	89	???		C9	CMP	IMM
0A	ASL	ACC	4A	LSR	ACC	8A	TXA		CA	DEX	
0B	???		4B	???		8B	???		CB	???	
0C	???		4C	JMP	ABS	8C	STY	ABS	CC	CPY	ABS
0D	ORA	ABS	4D	EOR	ABS	8D	STA	ABS	CD	CMP	ABS
0E	ASL	ABS	4E	LSR	ABS	8E	STX	ABS	CE	DEC	ABS
0F	???		4F	???		8F	???		CF	???	
10	BPL		50	BVC		90	BCC		D0	CNE	
11	ORA	(IND),Y	51	EOR	(IND),Y	91	STA	(IND),Y	D1	CMP	(IND),Y
12	???		52	???		92	???		D2	???	
13	???		53	???		93	???		D3	???	
14	???		54	???		94	STY	PAG0,X	D4	???	
15	ORA	PAG0,X	55	EOR	PAG0,X	95	STA	PAG0,X	D5	CMP	PAG0,X
16	ASL	PAG0,X	56	LSR	PAG0,X	96	STX	PAG0,Y	D6	DEC	PAG0,X
17	???		57	???		97	???		D7	???	
18	CLC		58	CLI		98	TYA		D8	CLD	
19	ORA	ABS,Y	59	EOR	ABS,Y	99	STA	ABS,Y	D9	CMP	ABS,Y
1A	???		5A	???		9A	TXS		DA	???	
1B	???		5B	???		9B	???		DB	???	
1C	???		5C	???		9C	???		DC	???	
1D	ORA	ABS,X	5D	EOR	ABS,X	9D	STA	ABS,X	DD	CMP	ABS,X
1E	ASL	ABS,X	5E	LSR	ABS,X	9E	???		DE	DEC	ABS,X
1F	???		5F	???		9F	???		DF	???	
20	JSR		60	RTS		A0	LDY	IMM	E0	CPX	IMM
21	AND	(IND,X)	61	ADC	(IND,X)	A1	LDA	(IND,X)	E1	SBC	(IND,X)
22	???		62	???		A2	LDX	IMM	E2	???	
23	???		63	???		A3	???		E3	???	
24	BIT	PAG0	64	???		A4	LDY	PAG0	E4	CPX	PAG0
25	AND	PAG0	65	ADC	PAG0	A5	LDA	PAG0	E5	SBC	PAG0
26	ROL	PAG0	66	ROR	PAG0	A6	LDX	PAG0	E6	INC	PAG0
27	???		67	???		A7	???		E7	???	
28	PLP		68	PLA		A8	TAY		E8	INX	
29	AND	IMM	69	ADC	IMM	A9	LDA	IMM	E9	SBC	IMM
2A	ROL	ACC	6A	ROR	ACC	AA	TAX		EA	NOP	
2B	???		6B	???		AB	???		EB	???	
2C	BIT	ABS	6C	JMP	IND	AC	LDY	ABS	EC	CPX	ABS
2D	AND	ABS	6D	ADC	ABS	AD	LDA	ABS	ED	SBC	ABS
2E	ROL	ABS	6E	ROR	ABS	AE	LDX	ABS	EE	INC	ABS
2F	???		6F	???		AF	???		EF	???	
30	BMI		70	BVS		B0	BCS		F0	BEQ	
31	AND	(IND),Y	71	ADC	(IND),Y	B1	LDA	(IND),Y	F1	SBC	(IND),Y
32	???		72	???		B2	???		F2	???	
33	???		73	???		B3	???		F3	???	
34	???		74	???		B4	LDY	PAG0,X	F4	???	
35	AND	PAG0,X	75	ADC	PAG0,X	B5	LDA	PAG0,X	F5	SBC	PAG0,X
36	ROL	PAG0,X	76	ROR	PAG0,X	B6	LDX	PAG0,Y	F6	INC	PAG0,X
37	???		77	???		B7	???		F7	???	
38	SEC		78	SEI		B8	CLV		F8	SED	
39	AND	ABS,Y	79	ADB	ABS,Y	B9	LDA	ABS,Y	F9	SBC	ABS,Y
3A	???		7A	???		BA	TSX		FA	???	
3B	???		7B	???		BB	???		FB	???	
3C	???		7C	???		BC	LDY	ABS,X	FC	???	
3D	AND	ABS,X	7D	ADC	ABS,X	BD	LDA	ABS,X	FD	SBC	ABS,X
3E	ROL	ABS,X	7E	ROR	ABS,X	BE	LDX	ABS,Y	FE	INC	ABS,X
3F	???		7F	???		BF	???		FF	???	

CAPITOLO 2

I FILE

2.1 COSA È UN FILE

Un file è un insieme di registrazioni, che, per un motivo qualunque, si desidera raggruppare.
Le registrazioni possono essere effettuate su un qualunque supporto fisico; i più comuni supporti fisici sono:

1) fogli di carta,
2) nastri di carta,
3) nastri magnetici,
4) dischi magnetici,
5) tamburi magnetici.

Sul supporto di tipo 1 le registrazioni avvengono sotto forma di caratteri direttamente leggibili, o disegni ottenuti per punti.
Sul supporto di tipo 2 le registrazioni avvengono sotto forma di fori, che rappresentano i caratteri in un codice. Se si è molto pazienti, tali registrazioni possono anche essere lette direttamente.
Sui supporti di tipo 3, 4 e 5 le registrazioni sono magnetiche e quindi non leggibili direttamente.

In questo libro ci occupiamo dei file, gestiti dal linguaggio BASIC, che si servono di supporti fisici del tipo 3, cassette magnetiche, e del tipo 4, floppy disk (dischetti).

Caratteristica di queste registrazioni è che si mantengono nel tempo a meno che non si intervenga su di esse in modo distruttivo.
Non consideriamo file le registrazioni nella memoria di lavoro del calcolatore, e, in particolare, nella memoria dedicata al video, in quanto hanno la caratteristica di essere temporanee. Abbiamo, però, visto che anche il video può essere gestito come un file.
È ovvio che tutto quello che diventa registrazione su un file, qualunque sia il supporto, transita dalla memoria del calcolatore.

I dati, presenti nella memoria del calcolatore, sono identificabili per mezzo dei nomi delle variabili che li contengono.

Le operazioni di SCRITTURA SU FILE da programma usano i nomi delle variabili per far uscire i dati o per prepararli ad uscire.

I dati, letti dall'esterno nella memoria del calcolatore, sono identificabili per mezzo dei nomi delle variabili che li ricevono. Le operazioni di LETTURA DA FILE usano i nomi delle variabili per far entrare i dati o per renderli disponibili dopo che sono entrati in memoria.

Gli elementi in gioco in questo tipo di operazioni sono quattro:

A) le VARIABILI che contengono o conterranno i dati,

B) il BUFFER, una zona di memoria di raccolta e transito per i dati,

C) il SUPPORTO fisico che contiene o conterrà le registrazioni,

D) il SOFTWARE DI GESTIONE.

Abbiamo già visto come viene gestito il BUFFER della stampante. Riepiloghiamo qui brevemente l'argomento.

- nella stampante è contenuta una zona di memoria RAM che viene usata come buffer,
- il programma manda dati alla stampante servendosi dei nomi delle variabili,
- il buffer via via si riempie,
- il buffer si svuota, cioè avviene fisicamente la stampa, quando è pieno oppure quando il programma invia un apposito carattere di controllo o chiude la trasmissione,
- il calcolatore invia i dati alla stampante carattere per carattere in codice ASCII,
- il software di gestione per l'invio dei dati sta nella ROM del calcolatore,
- il software necessario a trasformare i codici presenti nel buffer in caratteri stampati sta nella ROM della stampante.

Cose analoghe succedono con le periferiche, argomento della presente trattazione.

Per il registratore a nastro il BUFFER sta nella memoria RAM del calcolatore e il software necessario alla trasmissione sta nella ROM del calcolatore.

Il programma SCRIVE il contenuto delle variabili, i dati si ammucchiano nel buffer, il buffer viene fisicamente trasferito sul nastro (che si mette in movimento prima dell'inizio dell'operazione) quando esso è pieno, oppure quando il programma chiude la trasmissione. Il programma LEGGE i dati nelle variabili prelevandoli dal buffer, il buffer viene riempito alla prima lettura e tutte le volte che si cerca di leggere un carattere, dopo l'ultimo che esso contiene.

Le operazioni di lettura e scrittura del nastro avvengono in modo trasparente per l'utente; egli si accorge che sta avvenendo fisicamente un'operazione solo se guarda il nastro e lo vede in movimento.

Per l'unità a dischi, invece, il sistema operativo del calcolatore provvede solo all'invio dei dati o alla loro ricezione carattere per carattere (byte a byte) e alla gestione di una tabella relativa ai file aperti. Tale tabella contiene informazioni su tutti i file aperti sulle periferiche collegate al calcolatore. Il buffer e il software necessario alla gestione dei dischetti stanno rispettivamente nella RAM e nella ROM dell'unità a dischi. L'utente non si accorge quando i dati viaggiano tra il calcolatore e l'unità a dischi, mentre si accorge che sta avvenendo fisicamente un'operazione di lettura o scrittura dal fatto che si accende la spia rossa luminosa sull'unità. Il software necessario alla gestione delle operazioni disco prende il nome di DOS (Disk Operating System).

Riepiloghiamo nella figura che segue le relazioni che legano gli elementi sopra discussi.

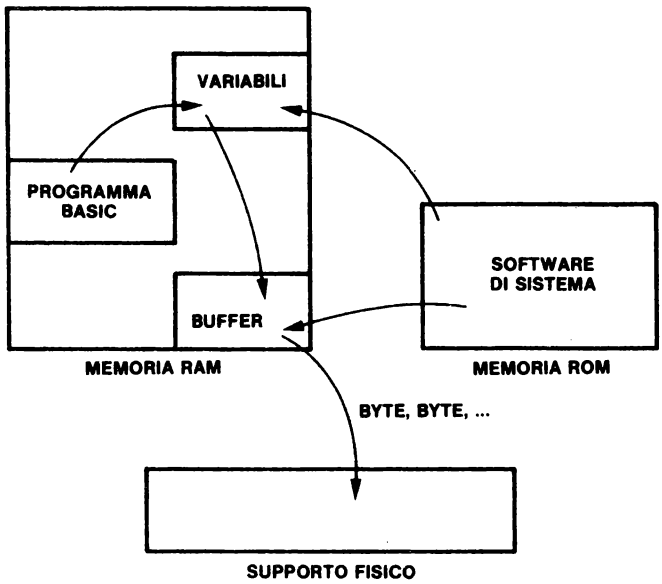


Figura 2.1 Software, variabili, buffer, supporto

In un file dobbiamo sempre distinguere l'aspetto fisico da quello logico. La STRUTTURA FISICA di un file dipende dal supporto di registrazione, dalla periferica e dal software di gestione a livello di linguaggio macchina (SOFTWARE DI BASE). La STRUTTURA LOGICA, che si sovrappone a una definita struttura fisica, dipende dal programma che crea il file. La struttura fisica in generale consiste nel fatto di potere e dovere scrivere un blocco fisico di byte di certe dimensioni. Per BLOCCO FISICO si intende la quantità di byte che viene fisicamente trasferita tra supporto e calcolatore, o viceversa, in una operazione di Input o Output. Per BLOCCO LOGICO si intende, invece, quel gruppo di dati che interessa il programma in un certo momento; il blocco logico può coincidere con un blocco fisico, costituirne una parte, o anche essere formato da più di un blocco fisico.

Vediamo ora la terminologia relativa all'aspetto LOGICO dei FILE. Le singole registrazioni, che si susseguono nel file e hanno un significato come gruppo di dati, si chiamano RECORD (blocco logico). In generale un record è composto da più dati, si dice che un record è composto da CAMPI (FIELD).

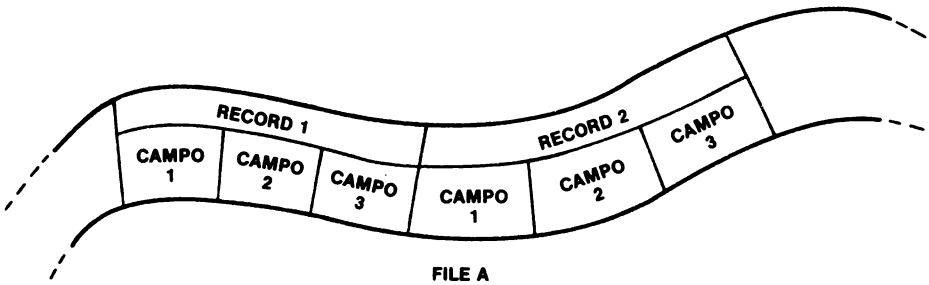


Figura 2.2 File, record, campi

Ogni campo è contraddistinto da un nome, per la logica del programma; come campo Cognome, campo Nome, campo Indirizzo, ecc.. All'interno del programma ad ogni campo corrisponde il nome di una variabile. Ogni file è contraddistinto da un nome, che lo individua. In generale al record non è attribuito un nome specifico, ma si può fare riferimento ad un record con il numero d'ordine con il quale esso compare nel file; primo record, secondo record, ecc.

Un file può essere formato da una serie di record, aventi tutti la stessa struttura, oppure da due o più strutture diverse di record che si ripetono con un certo ordine. Inoltre i record possono essere tutti della stessa lunghezza oppure di lunghezza variabile. I record di lunghezza fissa sono in generale formati da campi di lunghezza fissa. I record di lunghezza variabile possono essere formati da campi di lunghezza variabile, oppure da campi di lunghezza fissa, ma che si ripetono in numero diverso.

Nel primo volume, a proposito della stampante, non si è esplicitamente parlato di file. Vediamo ora come si può applicare la terminologia dei file alla stampa su carta.

Un qualunque tabulato è un file; ogni linea di stampa può essere considerata un record logico. Una riga di stampa (o il contenuto del buffer) può essere vista come un blocco fisico. Nella lista su carta di un programma BASIC ogni linea di programma può essere considerata un record logico di lunghezza variabile, formato da campi di lunghezza variabile. In un elenco ordinato di nomi e indirizzi ogni argomento può essere pensato come un record formato dai campi: cognome, nome, indirizzo, ecc.. Qualora il prospetto venga stampato ponendo sulla prima linea il cognome e nome e sulla seconda l'indirizzo, si può pensare a due tipi record che si ripetono in coppia.

Per ogni file si definisce il TIPO e il METODO DI ACCESSO.

Si definisce il TIPO di un file in dipendenza dal modo come il file viene costruito. Noi ci occupiamo dei tipi di file consentiti per il COMMODORE 64, in BASIC, e precisamente dei:

- File SEQUENZIALI,
- File RANDOM,
- File RELATIVI.

Un file può consistere in un insieme di dati, tipo gli elementi di un vettore di numeri, registrati uno dopo l'altro. Si tratta di una lista di dati e il file viene definito di TIPO SEQUENZIALE. Nell'esempio riportato ogni numero è un record del file, quindi il record è costituito da un solo campo. Caratteristica del file sequenziale è che ogni record, salvo il primo e l'ultimo, è preceduto e seguito da un altro record. Il file sequenziale si costruisce cominciando a scrivere il primo record e procedendo in sequenza.

Abbiamo un file di TIPO RANDOM quando i record vengono scritti facendo riferimento all'indirizzo del blocco fisico dove sono registrati. In realtà la terminologia, adottata nella letteratura della casa costruttrice, che nel seguito useremo anche noi, può NON DARE una chiara visione di questo tipo di file. A noi sembra che la definizione più corretta sia: FILE AD ACCESSO DIRETTO PER MEZZO DELL'INDIRIZZO FISICO. Approfondiremo l'argomento nel seguito chiarendo la relazione che sussiste, in questo caso, tra record logico e blocco fisico. I file random si costruiscono scrivendo i record come si vuole (nel rispetto delle regole del particolare file che si tratta), ma senza l'obbligo della sequenzialità, nel senso visto sopra.

Abbiamo un file DI TIPO RELATIVO quando i record sono scritti citando il loro numero d'ordine nel file: primo, secondo, quinto, ventesimo, terzo, ecc., senza l'obbligo di rispettare un ordine sequenziale. Riprendendo la definizione precedente possiamo definire questo tipo come: FILE AD ACCESSO DIRETTO PER MEZZO DELL'INDIRIZZO LOGICO.

Il tipo di un file viene definito al momento della sua creazione e non è modificabile.

Per METODO DI ACCESSO si intende il modo nel quale si può accedere ai record che compongono il file. I metodi di accesso disponibili sono due:

- Accesso SEQUENZIALE,
- Accesso DIRETTO.

Il metodo di accesso sequenziale consiste nella lettura dei record uno dopo l'altro, partendo dal primo; esso si può usare per tutti i tre tipi di file sopra descritti. Il metodo di accesso diretto, che consiste nella lettura di un ben determinato record, indipendentemente dalla lettura precedente, si può usare solo per i file random e per i file relativi.
Sulla cassetta è possibile solo la gestione dei file sequenziali. Sui dischetti sono gestibili i tre tipi di file.

Le registrazioni magnetiche avvengono un byte alla volta, si ha la trasmissione di sequenze di byte. Ogni byte contiene un codice numerico che può variare da 0 a 255, e che viene abitualmente definito carattere. Anche i dati numerici, che all'interno della memoria del calcolatore sono memorizzati come numeri interi o floating-point, sono trasformati in uscita in stringhe di caratteri. Una trasformazione inversa avviene in fase di lettura dal supporto magnetico verso la memoria del calcolatore. Ogni stringa di caratteri, che rappresenta un dato, deve terminare con un carattere separatore riconoscibile dal sistema.

2.2 OPERAZIONI PER LA GESTIONE DEI FILE

Il programma BASIC per gestire un file deve svolgere le seguenti operazioni:

1) Aprire la comunicazione con il file,

2) Leggere o scrivere il file,

3) Chiudere la comunicazione con il file.

L'operazione 1) stabilisce la comunicazione con la periferica e con un determinato file, per effettuare un certo tipo di operazione. Si tratta dell'operazione OPEN; la parola chiave è seguita da alcuni parametri, che analizziamo separatamente per la cassetta e per il dischetto nei due capitoli seguenti. Dopo la OPEN i dati relativi al file aperto sono memorizzati nella RAM del calcolatore nella tabella per la gestione dei file. Il byte 152 (0098H) contiene il numero dei file aperti (LDTND), puntatore alla tabella dei file. La tabella di gestione dei file si trova a partire dal byte 601 (0259H); essa è formata da tre vettori di 10 byte ciascuno. La struttura è la seguente:

- LAT da 601 a 610 (0259H-0262H) per i numeri logici dei file attivi.
- FAT da 611 a 620 (0263H-026CH) per i numeri logici delle periferiche dei file attivi.
- SAT da 621 a 630 (026DH-0276H) per gli indirizzi secondari dei file attivi.

I tre byte 601, 611 e 621 contengono le informazioni relative al primo file, e gli altri, analogamente in gruppi di tre, per gli altri file. Dalle dimensioni della tabella sembra che si possano aprire contemporaneamente 10 file. In realtà vedremo che non è così.

L'operazione 2) legge o scrive gruppi di dati trasferendoli dal o nel buffer e, certe volte, dà luogo anche alla lettura o scrittura fisica nel o dal buffer.
Anche questa operazione viene trattata diffusamente nel seguito; sono le istruzioni INPUT#, GET# e PRINT#.

L'operazione 3) chiude la comunicazione con il file e la periferica; si tratta della CLOSE. Essa elimina il file dalla tabella di gestione, e, in certi casi, dà luogo all'ultima registrazione sul supporto.

2.3 COME SI ELABORANO I FILE DI DATI

Le operazioni che si eseguono sui file di dati possono essere dei tipi seguenti:

1) Creazione ex-novo del file.
2) Aggiornamento del file.
3) Ricerca di dati sul file.

Il modo con il quale possono avvenire queste operazioni dipende dal tipo del file, definito nella fase di creazione.

CREAZIONE EX-NOVO DEL FILE

Consiste nella scrittura iniziale del file. In questa fase viene assegnato un nome al file, ne viene stabilito il tipo, sono definite le caratteristiche dei record e dei campi.

Nei due capitoli seguenti precisiamo come viene svolta la creazione del file per ogni tipo preso in esame.

AGGIORNAMENTO DEL FILE

Consiste nella modifica di record già presenti nel file, nella cancellazione di qualche record e nell'aggiunta di nuovi record. L'operazione si svolge con modalità diverse in dipendenza dal tipo del file e dal supporto fisico usato.

RICERCA DI DATI SUL FILE

Possiamo raggruppare sotto questo nome tutte le operazioni di utilizzo dei dati registrati in un file. La più semplice e banale è la lista su carta di tutti i record del file. Altre operazioni possono essere di tipo selettivo con riferimento al contenuto di uno specifico campo. Un'operazione molto comune è la riorganizzazione dei record di un file in base al contenuto di un campo; essa prende il nome di SELEZIONE (SORTING) e richiede una notevole quantità di memoria, sia centrale che di massa. Le modalità di ricerca consentite su un file dipendono dal suo tipo, dal supporto fisico e dal software a disposizione.

Il problema principale inerente alla ricerca di un record in un file è quello di sapere quale record si vuole. Consideriamo, come esempio, un file relativo ad alcune informazioni sui dodici mesi dell'anno. Questo file è formato da 12 record, uno per ogni mese, e risulta ovvio che il primo record è quello di gennaio, l'ultimo quello di dicembre, il decimo quello di ottobre e simili. Consideriamo ora un file di indirizzi, che contiene 500 argomenti; come facciamo a sapere che il record relativo al signor Bianchi Carlo è, per esempio, il dodicesimo? Un modo potrebbe essere avere a disposizione un elenco su carta, dove ogni nome, riportato in ordine alfabetico per comodità di consultazione, sia accompagnato da un numero, il numero d'ordine del record. Abbiamo ora introdotto il concetto di INDICE di un file, anche se su carta. Se non possediamo alcun indice, per trovare il record del signore in questione, possiamo solo cominciare a leggere il file dal primo record controllando i nomi che si incontrano, fino a trovare quello cercato. Questo metodo è, per esempio, l'unico a disposizione per i file su nastro, che sono di tipo sequenziale.

CAPITOLO 3

FILE SU CASSETTA

3.1 INTRODUZIONE

Al file su cassetta può essere assegnato un nome di riconoscimento; esso viene registrato insieme ad alcuni caratteri di controllo, generati dal sistema. L'insieme di questi dati costituisce il blocco di testata del file sul nastro. Se l'utente registra senza nome, il blocco di testata contiene solo i caratteri di controllo. Questi caratteri di controllo risultano trasparenti per l'utente nelle normali operazioni di gestione dei file da BASIC. Ti consigliamo di registrare usando sempre un nome di riconoscimento; esso permette infatti di distinguere tra loro le diverse registrazioni, che possono trovarsi sulla stessa cassetta, e di richiamarle in modo selettivo.

Il COMMODORE 64 consente di gestire due tipi di file su nastro; si tratta sempre di FILE SEQUENZIALI, ma le modalità di registrazione sono diverse. Si possono avere:

- I file ASCII: in essi i dati sono registrati carattere per carattere in blocchi separati uno dall'altro da un CARATTERE SEPARATORE.

- I file BINARI: in essi i caratteri sono registrati in modo apparentemente continuo.

I file ASCII sono usati per registrare dati, i file BINARI per registrare programmi. I due tipi di file sono gestiti con istruzioni diverse; nel secondo Volume abbiamo già trattato i file di programmi.

La differenza, dal punto di vista della gestione, tra i due tipi di file sta nel fatto che, mentre i file ASCII possono essere letti a pezzi e il nastro viene avviato per leggere un blocco e poi fermato, i file BINARI possono solo essere letti tutti in una sola volta; il loro contenuto viene cioè trasferito completamente in memoria a partire da

un certo byte. Inoltre, se l'indirizzo del byte dal quale inizia la memorizzazione non coincide con quello registrato nella testata del file, avviene la modifica dei due byte di LINK di ogni linea di programma (rilocazione). Questo succede in quanto i file binari possono essere solo file di programmi BASIC.

Il buffer per le operazioni nastro si trova nella RAM del calcolatore a partire dal byte 828 (033CH) e occupa 192 byte, in conseguenza il blocco fisico di nastro che viene letto o scritto è di 192 caratteri. Il sistema registra due volte ogni blocco fisico su nastro e, in fase di lettura, confronta la prima lettura con la seconda. Se le due letture non coincidono viene segnalato un errore. L'utente non si accorge di questa doppia registrazione, può solo notare che le operazioni nastro sono lente. Tra un blocco fisico di registrazione e il successivo si trova un pezzetto di nastro non utilizzato (GAP); esso è necessario per consentire la fermata e il riavvio del nastro. Ogni file è composto almeno da tre blocchi fisici: la testata, il corpo del file e il blocco di chiusura.

Altri indirizzi importanti per la gestione delle operazioni su cassetta sono:

FBUFPT, byte 113/114 (0071H-0072H)
puntatore al buffer della cassetta.

STATUS, byte 144 (0090H)
parola di stato per le routine KERNAL di Input/Output, nel programma BASIC viene richiamata con ST.

SVXT, byte 146 (0092H)
costante di controllo per la velocità di scorrimento del nastro.

VERCK, byte 147 (0093H)
indicatore: 0=LOAD, 1=VERIFY.

SYNO, byte 150 (0096H)
numero per sincronizzazione cassetta.

RTY, byte 155 (009BH)
carattere di parità del nastro.

DPSW, byte 156 (009CH)
indicatore ricezione byte da nastro.

PTR1, byte 158 (009EH)
indicatore errore passo 1 (primo blocco) nastro.

PTR2, byte 159 (009FH)
indicatore errore passo 2 (secondo blocco) nastro.

CNTDN, byte 165 (00A5H)
contatore in decremento per sincronizzazione cassetta.

BUFPNT, byte 166 (00A6H)
puntatore al buffer del nastro.

INBIT, byte 167 (00A7H)
bit di input cassetta.

BITC1, byte 168 (00A8H)
contatore bit input cassetta.

RIDATA, byte 170 (00AAH)
byte di input cassetta.

RIPRTY, byte 171 (00ABH)
contatore corto cassetta.

SAL, byte 172/173 (00ACH-00ADH)
indirizzo iniziale memorizzazione file binari.

EAL, byte 174/175 (00AEH-00AFH)
indirizzo finale memorizzazione file binari.

CMPO, byte 176/177 (00B0H-00B1H)
costanti per operazioni nastro.

TAPE1, byte 178/179 (00B2H-00B3H)
puntatore buffer nastro.

BITTS, byte 180 (00B4H)
contatore bit output cassetta.

NXTBIT, byte 181 (00B5H)
prossimo bit da inviare, indicatore fine nastro.

FNLEN, byte 183 (00B7H)
lunghezza nome file corrente.

LA, byte 184 (00B8H)
numero logico file corrente.

SA, byte 185 (00B9H)
indirizzo secondario file corrente.

FA, byte 186 (00BAH)
numero logico periferica corrente.

FNADR, byte 187/188 (00BBH-00BCH)
puntatore al nome del file corrente.

ROPRTY, byte 189 (00BDH)
parità output cassetta.

FSBLK, byte 190 (00BEH)
contatore blocco Input/Output cassetta.

CAS1, byte 192 (00C0H)
arresto motore nastro.

MEMUSS, byte 195/196 (00C3H-00C4H)
memoria temporanea LOAD nastro.

IRQTMP, byte 671/672 (029FH-02A0H)
contiene il vettore IRQ durante Input/Output da nastro.

Byte 674 (02A2H)
controllo sensore cassetta durante Input/Output da nastro.

Byte 675 (02A3H)
memoria temporanea per lettura nastro.

Byte 676 (02A4H)
indicatore temporaneo IRQ durante lettura nastro.

Puoi analizzare il contenuto di alcuni di questi byte in diversi momenti, usando la funzione PEEK, per renderti conto meglio del loro significato.

3.2 FILE DI DATI

Su nastro si possono gestire solo file sequenziali. Il file può essere composto da record di lunghezza fissa o variabile. Il record può essere composto da un solo campo o da più campi di lunghezza fissa o variabile. Il fatto di gestire campi di lunghezza variabile (e in conseguenza record di lunghezza variabile), dipende dalla logica del programma. Per la logica di costruzione del file un campo termina con un carattere separatore valido, ma al limite, e con le limitazioni che vedremo, un intero file potrebbe essere costituito da un solo campo senza caratteri separatori.

Esponiamo per prima cosa le istruzioni BASIC disponibili per la gestione dei file su cassetta, facendo seguire alcuni esempi.

OPEN lfn,dn,sa,fn

si deve usare per aprire un file su nastro, ponendo: dn=1 o omettendolo, dato che viene assunto per default uguale a uno. Per omettere un parametro, qualora ne seguano altri, si devono scrivere due virgole vicine.

Il parametro sa può avere i seguenti valori:

sa=0 o mancante (valore di default zero) per operazione di lettura,
sa=1 per operazione di scrittura senza registrazione del carattere di fine-nastro (END-OF-TAPE) dopo la chiusura del file,
sa=2 per operazioni di scrittura con registrazione del carattere di fine-nastro dopo la chiusura del file.

Il parametro lfn (logical-file-number) può avere un valore compreso tra 1 e 255, ma di solito si usano numeri inferiori a 127; tale numero deve essere sempre citato nelle istruzioni relative al file.
Il nome del file, fn, deve essere una stringa di al massimo 16 caratteri; esso può essere passato come costante o come variabile. Tale nome può essere omesso, ma la cosa non è consigliabile.
Tutti gli altri parametri che compaiono nell'istruzione possono essere costanti o variabili numeriche.

L'istruzione OPEN provoca la scrittura del blocco di testata del file, tale blocco transita dal buffer, come vedremo dagli esempi che seguono.

PRINT#lfn,lista

provoca la scrittura dei dati della lista sul file contraddistinto dal numero logico lfn. Nel caso della cassetta può essere ovviamente trattato un file per volta, ma se lo lfn

usato nella PRINT non corrisponde a quello usato nella OPEN si ha segnalazione di errore. Analogamente si ha segnalazione di errore se il file non è stato aperto per scrivere. I dati della lista vengono trasferiti nel buffer della cassetta; quando il buffer è pieno, esso viene scritto su nastro. Può succedere che l'ultimo dato inviato, quello che finisce di riempire il buffer, venga spezzato in modo che i primi caratteri sono scritti fisicamente in un blocco e gli altri nel successivo. Tu comunque non ti accorgi di questo nè in fase di scrittura, nè in fase di lettura.

Come sempre, la lista di dati può essere formata da costanti o da variabili separate tra loro da virgola o da punto e virgola. Il carattere separatore virgola provoca l'aggiunta di spazi ai caratteri del dato che lo precede, il carattere separatore punto e virgola non provoca l'aggiunta di spazi. In sostanza questi caratteri separatori agiscono come sul video, ma non provocano la registrazione di un carattere separatore. Per poter rileggere poi con l'istruzione INPUT#, che legge in variabili, è necessario che i dati siano registrati facendo comparire dopo ognuno di essi un carattere separatore valido, cioè riconosciuto come tale dal sistema. I caratteri separatori riconosciuti sono il RETURN (CHR$(13)) e la virgola (CHR$(44)). Il primo può essere passato usando la funzione CHR$(13) oppure facendo terminare la lista di dati senza punteggiatura. Il secondo può essere passato usando la funzione CHR$(44) oppure come stringa costante ",". Abbiamo però delle limitazioni nell'uso dei due caratteri separatori, infatti il sistema non consente che i dati, qualora debbano poi essere letti con INPUT#, siano registrati in modo che tra due caratteri CHR$(13) siano compresi più di 79 caratteri. Inoltre, se si usa come carattere separatore registrato la virgola, l'istruzione INPUT# deve leggere riportando nella lista dei dati tanti nomi di variabili quanti sono i dati compresi tra due CHR$(13); se non si opera così, si perdono dei dati. Se un dato stringa deve contenere il carattere virgola o il carattere punto e virgola, è necessario forzare le virgolette all'inizio e alla fine con CHR$(34).

In sostanza possiamo affermare che:

- per separare i record logici si deve usare il carattere separatore CHR$(13), mentre per separare i campi si può usare anche il carattere CHR$(44), purchè la somma dei caratteri non superi 79, comprese le virgole separatrici, e i campi vengano letti tutti da una sola INPUT#;
- per poter leggere con INPUT# il numero dei caratteri compresi tra due CHR$(13) non deve superare 79;
- quando la lista dei dati termina senza punteggiatura il sistema aggiunge un CHR$(13);
- può essere usato sempre come carattere separatore CHR$(13) sia a livello di campo che di record, e con esso si può leggere con maggior libertà;
- i dati vengono scritti carattere per carattere trasformando i numeri in stringa con le regole note;

• i dati, scritti senza caratteri separatori registrati o con caratteri separatori registrati, possono essere letti carattere per carattere con l'istruzione GET#:

INPUT#lfn,lista

legge dal file, aperto per leggere con numero logico lfn, i dati trasferendoli nelle variabili della lista. Un dato viene considerato terminato quando viene incontrato il carattere CHR$(13) o il carattere CHR$(44). Se un dato supera il numero dei caratteri consentiti si ha segnalazione di errore. Se si cerca di leggere in una variabile numerica un dato non numerico si ha segnalazione di errore. Se la lista contiene un numero di variabili inferiore al numero dei dati compresi tra due RETURN e separati da virgole, i dati in più non vengono letti.

GET#lfn,lista

legge dal file, aperto per leggere con numero logico lfn, un carattere per volta. Dopo l'operazione ogni variabile della lista contiene un carattere. Ti consigliamo di usare nella lista solo variabili stringa, per evitare errori.
Con GET# si può leggere un file scritto senza usare mai caratteri separatori validi tra i dati; deve, però, pensare poi il programma a interpretare opportunamente l'insieme dei caratteri letti.

CLOSE lfn

chiude il file aperto con numero logico lfn.
Segue il programma PRFL1C, come primo esempio.

```
10 REM PRFL1C
15 REM CREAZIONE FILE SEQUENZIALE
20 REM RECORD LOGICO FORMATO DA 3 CAMPI
25 REM CAMPI DI LUNGHEZZA VARIABILE
30 REM A$←COGNOME
35 REM B$←NOME
40 REM C$←DATO NUMERICO LETTO COME STRINGA
45 REM SEPARATORE DI CAMPO = CHR$(13)
50 REM UN SOLO PRINT SCRIVE IL RECORD
55 REM SENZA PUNTEGGIATURA FINALE
60 REM IL SISTEMA AGGIUNGE CHR$(13)
65 REM ALLA FINE DEL RECORD
70 REM ";" COME SEPARATORE TRA LE VARIABILI
75 REM PER NON AGGIUNGERE SPAZI
80 CH$=CHR$(13):OPEN1,1,1,"FL1C"
```

```
85 V$=CHR$(44):PV$=CHR$(59)
90 CD$="▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
95 PRINT"▒":INPUT"COGNOME: ";A$
100 INPUT"NOME: ";B$
105 INPUT"NUMERO:";C$
110 PRINT#1,A$;CH$;B$;CH$;C$
115 PRINT#1,A$,CH$,B$,CH$,C$
120 PRINT#1,A$;V$;B$;V$;C$
125 PRINT#1,A$;PV$;B$;PV$;C$
130 R$="":PRINT CD$;"ALTRO RECORD (S/N)?";
135 GETR$:IFR$=""THEN135
140 IFR$<>"S"THEN150
145 GOTO95
150 PRINT"▒RIAVV. IL NASTRO-POI SCRIVI CONT"
155 CLOSE1:STOP
160 OPEN1,1,0,"FL1C":OPEN4,4
165 PRINT#4,"FILE LETTO CON GET#":PRINT#4
170 GET#1,R$:TS=ST:PRINT#4,R$;"*";
175 IFTS<>64THEN170
180 PRINT#4:CLOSE1:CLOSE4
185 PRINT"▒RIAVV. IL NASTRO-POI SCRIVI CONT"
190 STOP
195 OPEN1,1,0,"FL1C":OPEN4,4
200 PRINT#4,"FILE LETTO CON INPUT#":PRINT#4
205 INPUT#1,R$:TS=ST:PRINT#4,R$
210 IFTS<>64THEN205
215 PRINT#4:CLOSE1:CLOSE4:STOP
```

Il programma PRFL1C fa le seguenti cose:

- crea un file sequenziale su nastro di nome FL1C (linea 80),
- il record logico del file è formato da 3 campi di lunghezza variabile, e per essi non viene controllata la lunghezza: cognome, nome e numero (linee 95-105),
- i 3 campi sono trattati come stringhe,
- ogni record logico viene scritto 4 volte:

1).usando come carattere separatore tra le variabili il punto e virgola e come carattere separatore registrato il CHR$(13) (linea 110),

2).come in 1), ma usando come carattere separatore tra le variabili la virgola (linea 115),

3).usando come carattere separatore tra le variabili il punto e virgola e come carattere separatore registrato la virgola (CHR$(44)) (linea 120),

4).come in 3), ma usando come carattere separatore registrato il punto e virgola (CHR$(59)) (linea 125),

- dopo aver scritto 4 volte un record viene chiesto se si vuole aggiungere un altro record (un gruppo di 4) (linee 130-145),
- quando la risposta è NO, viene chiuso il file (linea 155),
- viene chiesto di riavvolgere il nastro (linea 150) e viene aperto il file per lettura (linea 160),
- il file viene letto con GET# e stampato carattere per carattere (linee 170-180),
- viene chiesto di riavvolgere il nastro (linea 185) e viene aperto nuovamente il file per lettura (linea 195),
- il file viene letto con INPUT# e stampato (linee 200-215).

Seguono i risultati di FL1C, nel caso si siano caricati 2 record diversi, contenenti i seguenti dati:

Cognome: PRIMOC, Nome: PRIMON, Numero: 11111
Cognome: SECONDOC, Nome: SECONDON, Numero: 22222

```
FILE LETTO CON GET#

P*R*I*M*O*C*
*P*R*I*M*O*N*
*1*1*1*1*1*
*P*R*I*M*O*C* * * * * * * * * * *
* * * * * * * * * * *P*R*I*M*O*N* * * * * * * * * * *
* * * * * * * * * * *1*1*1*1*1*
*P*R*I*M*O*C*,*P*R*I*M*O*N*,*1*1*1*1*1*
*P*R*I*M*O*C*;*P*R*I*M*O*N*;*1*1*1*1*1*
*S*E*C*O*N*D*O*C*
*S*E*C*O*N*D*O*N*
*2*2*2*2*2*
*S*E*C*O*N*D*O*C* * * * * * * * * * *
*·* * * * * * * * * *S*E*C*O*N*D*O*N* * * * * * * * * * *
* *,* * * * * * * * *2*2*2*2*2*
*S*E*C*O*N*D*O*C*,*S*E*C*O*N*D*O*N*,*2*2*2*2*2*
*S*E*C*O*N*D*O*C*;*S*E*C*O*N*D*O*N*;*2*2*2*2*2*
*
```

```
FILE LETTO CON INPUT#

PRIMOC
PRIMON
11111
PRIMOC
PRIMON
11111
PRIMOC
PRIMOC;PRIMON;11111
SECONDOC
SECONDON
22222
SECONDOC
SECONDON
22222
SECONDOC
SECONDOC;SECONDON;22222
```

Per mettere meglio in evidenza i risultati della lettura con GET#, ogni carattere viene stampato facendolo seguire dal carattere "*". Come puoi vedere:

- il carattere separatore ";" tra le variabili non aggiunge spazi,
- il carattere separatore "," tra le variabili aggiunge 10 spazi dopo il dato,
- dopo il CH$ (che è il carattere RETURN), la stampa continua alla riga seguente e ci sono 10 spazi, il primo asterisco è quello che viene dopo CH$,
- i caratteri separatori registrati "," e ";" sono letti e stampati,
- ogni istruzione di PRINT# (linee 110-125) termina senza punteggiatura e quindi il sistema aggiunge il RETURN.

La seconda parte dei risultati riguarda la lettura con INPUT#. Come puoi vedere:

- non si nota differenza nella lettura dei primi due record, infatti gli spazi a sinistra dei dati vengono persi e quelli a destra non si vedono (ma ci sono),
- per il terzo record la virgola separatrice registrata fa perdere gli ultimi due campi, dato che la lista dati della INPUT# contiene una sola variabile,
- per il quarto record il punto e virgola separatore registrato non viene riconosciuto e i tre campi vengono letti come un campo unico.

Puoi sostituire alla linea 205 del programma le due linee che seguono:

```
205 INPUT#1,R1$,R2$,R3$:TS=ST
206 PRINT#4,R1$:PRINT#4,R2$:PRINT#4,R3$
```

ottenendo di non perdere i tre campi quando usi il separatore registrato virgola. Infatti in questo caso la linea 205 legge tutto il record in una sola volta; la lista dati è formata da tre variabili.

Puoi provare il programma PRFL1C rispondendo alla richiesta dati con campi abbastanza lunghi, tipo 60 caratteri; vedrai che alla INPUT# di linea 205 otterrai il messaggio di errore STRING TOO LONG, infatti con l'aggiunta degli spazi si superano i 79 caratteri.

Segue il programma PRFL2C.

```
10 REM PRFL2C
15 REM CARICAMENTO FILE NUMERICO
20 REM OGNI RECORD LOGICO COMPRENDE 2 CAMPI
25 REM IL PRIMO E' UN NUMERO INTERO
30 REM IL SECONDO UN NUMERO DECIMALE
35 REM SI USA CHR$(13) COME SEPARATORE DI CAMPO
40 REM PER USCIRE DARE UN INTERO NULLO
45 REM IL RETURN FINALE LO METTE IL SISTEMA
50 CH$=CHR$(13)
55 OPEN1,1,1,"FL2C":OPEN4,4
60 PRINT"NUMERI INTERI <= 32767"
65 PRINT"PER USCIRE RISPONDERE 0 (ZERO) A INTERO"
70 INPUT"INTERO: ";I%
75 IFI%=0THENCLOSE1:GOTO95
80 INPUT"DECIMALE: ";D
85 PRINT#1,I%;CH$;D
90 GOTO70
95 Z$="DOPO OPERAZIONE CLOSE":GOSUB185
100 PRINT"RIAVV. IL NASTRO-POI SCRIVI CONT"
105 STOP
110 REM LETTURA FILE NUMERICO
115 REM CON INPUT VENGONO LETTI I 2 CAMPI
120 REM ESSI VENGONO STAMPATI
```

```
125 REM QUANDO E' FINITO IL FILE VIENE LETTO
130 REM E STAMPATO IL CONTENUTO DEL BUFFER
135 OPEN1,1,0,"FL2C"
140 Z$="DOPO OPERAZIONE OPEN":GOSUB185
145 PRINT#4:PRINT#4,"CONTENUTO FILE NUMERICO"
150 PRINT#4:PRINT#4
155 INPUT#1,I%,D:TS=ST
160 PRINT#4,I%,D
165 IFTS<>64THEN155
170 Z$="DOPO LETTURA DATI NUMERICI":GOSUB185
175 CLOSE1
180 CLOSE4:STOP
185 PRINT#4:PRINT#4,Z$
190 PRINT#4,"CONTENUTO BUFFER CASSETTA"
195 PRINT#4
200 I=828
205 FORK=1TO18:FORL=1TO4
210 PRINT#4,I;" ";MID$(STR$(PEEK(I))+"  ",2,3);" ";
215 I=I+1:NEXTL:PRINT#4
220 NEXTK:PRINT#4:RETURN
```

Il programma PRFL2C fa le seguenti cose:

- crea un file sequenziale il cui record logico comprende due campi numerici: un numero intero e un numero decimale (con cifre dopo il punto decimale) (linee 55-90),
- viene usato come carattere separatore sia di campo che di record il RETURN (linea 85),
- dopo la chiusura del file viene stampato il contenuto del buffer della cassetta (linea 95, sottoprogramma 185-220),
- viene chiesto di riavvolgere il nastro (linea 100) e poi viene aperto il file per leggere (linea 135),
- dopo l'apertura del file viene stampato il contenuto del bufffer (linea 140),
- viene letto e stampato il file (linee 145-165),
- viene stampato il contenuto del buffer dopo la lettura del file (linea 170).

Seguono i risultati nel caso si siano caricate nel file le seguenti 4 coppie di numeri:

123, -678.43
-67, 543.78
3456, 90.67
-43, 890.5

DOPO OPERAZIONE CLOSE
CONTENUTO BUFFER CASSETTA

828	2	829	32	830	49	831	50
832	51	833	32	834	13	835	45
836	54	837	55	838	56	839	46
840	52	841	51	842	32	843	13
844	45	845	54	846	55	847	32
848	13	849	32	850	53	851	52
852	51	853	46	854	55	855	56
856	32	857	13	858	32	859	51
860	52	861	53	862	54	863	32
864	13	865	32	866	57	867	48
868	46	869	54	870	55	871	32
872	13	873	45	874	52	875	51
876	32	877	13	878	32	879	56
880	57	881	48	882	46	883	53
884	32	885	13	886	0	887	32
888	32	889	32	890	32	891	32
892	32	893	32	894	32	895	32
896	32	897	32	898	32	899	32

DOPO OPERAZIONE OPEN
CONTENUTO BUFFER CASSETTA

828	4	829	60	830	3	831	80
832	12	833	70	834	76	835	50
836	67	837	32	838	32	839	32
840	32	841	32	842	32	843	32

844	32	845	32	846	32	847	32
848	32	849	32	850	32	851	32
852	32	853	32	854	32	855	32
856	32	857	32	858	32	859	32
860	32	861	32	862	32	863	32
864	32	865	32	866	32	867	32
868	32	869	32	870	32	871	32
872	32	873	32	874	32	875	32
876	32	877	32	878	32	879	32
880	32	881	32	882	32	883	32
884	32	885	32	886	32	887	32
888	32	889	32	890	32	891	32
892	32	893	32	894	32	895	32
896	32	897	32	898	32	899	32

CONTENUTO FILE NUMERICO

123	-678.43
-67	543.78
3456	90.67
-43	890.5

DOPO LETTURA DATI NUMERICI
CONTENUTO BUFFER CASSETTA

828	2	829	32	830	49	831	50
832	51	833	32	834	13	835	45
836	54	837	55	838	56	839	46
840	52	841	51	842	32	843	13
844	45	845	54	846	55	847	32
848	13	849	32	850	53	851	52
852	51	853	46	854	55	855	56
856	32	857	13	858	32	859	51
860	52	861	53	862	54	863	32
864	13	865	32	866	57	867	48
868	46	869	54	870	55	871	32
872	13	873	45	874	52	875	51
876	32	877	13	878	32	879	56
880	57	881	48	882	46	883	53

```
884  32   885  13   886  0    887  32
888  32   889  32   890  32   891  32
892  32   893  32   894  32   895  32
896  32   897  32   898  32   899  32
```

Come puoi vedere, il contenuto del buffer dopo la CLOSE che chiude la creazione del file è uguale a quello stampato dopo la lettura del file. Trovi come primo carattere 2, che significa file di dati. Dopo trovi i numeri caricati, carattere per carattere in codice ASCII, preceduti da uno spazio (32) o dal segno meno (45), dopo il numero si ha ancora uno spazio (32) e poi il carattere separatore RETURN (13). Dopo l'ultimo numero, nel byte 886 puoi vedere lo zero binario di chiusura del file. Il contenuto del buffer dopo l'operazione di OPEN per leggere ci mostra 4 nel primo byte, e il nome del file FL2C nei byte da 833 a 836.

3.3 FILE DI PROGRAMMI

Nel secondo Volume, abbiamo visto i comandi relativi alla memorizzazione e al caricamento dei programmi su nastro. In questo paragrafo analizziamo il contenuto dei file di programmi, cercando di leggerli come file sequenziali ASCII.

Abbiamo preso il programma PRGINMEM che segue e l'abbiamo salvato su cassetta con: SAVE "PRGINMEM". Tale programma stampa se stesso a scopo dimostrativo.

```
1 REM PRGINMEM
5 OPEN4,4:CMD4:PRINT"PROGRAMMA IN MEMORIA":PRINT
10 X=256*PEEK(44)+PEEK(43)
15 Y1=PEEK(X):Y2=PEEK(X+1)
17 IFY1+Y2=0THENPRINTX;"**";Y1;Y2:GOTO60
20 PRINTX;"**";Y1;Y2;"LINK=";Y2*256+Y1
25 Y1=PEEK(X+2):Y2=PEEK(X+3)
30 PRINT"NUMERO LINEA: ";Y2*256+Y1
35 X=X+4:C=0
37 Y1=PEEK(X):IFY1=0THEN55
40 PRINTY1;"  .";:X=X+1:C=C+1
45 IFC=6THENPRINT:C=0
50 GOTO37
55 PRINTY1:X=X+1:GOTO15
60 PRINT#4:CLOSE4:STOP
```

Il programma FILEPROG1, che segue, legge tale file aprendolo come file sequenziale ASCII.

```
10 REM FILEPROG1
15 GOSUB145:REM AZZERA BUFFER
20 OPEN1,1,0,"PRGINMEM"
25 OPEN4,4
30 PRINT#4,"BUFFER DOPO OPEN PRGINMEM"
35 PRINT#4:GOSUB100:REM LISTA BUFFER
40 GOSUB145:REM AZZERA BUFFER
45 GET#1,A$:REM LEGGE UN CARATTERE
50 PRINT#4:PRINT#4,"BUFFER DOPO UN GET"
55 PRINT#4:GOSUB100:REM LISTA BUFFER
60 REM CICLO 191 GET PER ARRIVARE ALLA
65 REM FINE DEL BUFFER
70 FORK=1TO191:GET#1,A$
75 NEXTK
80 GET#1,A$:REM UN GET DOPO I PRIMI 192
85 PRINT#4:PRINT#4,"BUFFER DOPO 193 GET"
90 PRINT#4:GOSUB100:REM LISTA BUFFER
95 CLOSE1:CLOSE4:STOP
100 REM LISTA BUFFCASS
105 REM INDIRIZZI BUFFER CASSETTA
110 I1=828:I2=1019
115 FORI=I1TOI2STEP6
120 FORK=0TO5
125 PRINT#4,PEEK(I+K);" ";
130 NEXTK
135 PRINT#4:NEXTI
140 RETURN
145 REM AZZERA BUFFCASS
150 I1=828:I2=1019
155 FORI=I1TOI2
160 POKEI,32
165 NEXTI
170 RETURN
```

Il programma FILEPROG1 fa le seguenti cose:

- pone il carattere spazio (32) in tutte le posizioni del buffer (linea 15, sottoprogramma 145-170),
- apre il file PRGINMEM come file da leggere (linea 20),

• stampa il contenuto del buffer dopo la OPEN (linee 30-35, sottoprogramma 100-140),

• pulisce il buffer (linea 40),

• legge con GET# un carattere dal file, provocando il riempimento del buffer (linea 45),

• stampa il contenuto del buffer (linee 50-55),

• esegue un ciclo di 191 GET#, per arrivare alla fine del buffer (linee 60-75),

• esegue un nuovo GET# per provocare un ulteriore riempimento del buffer (linea 80),

• stampa il contenuto del buffer dopo questo nuovo GET# (linee 85-90),

• chiude il file (linea 95).

Seguono i risultati.

```
BUFFER DOPO OPEN PRGINMEM

 1    1    8    82    9    80
 82   71   73   78   77   69
 77   32   32   32   32   32
 32   32   32   32   32   32
 32   32   32   32   32   32
 32   32   32   32   32   32
 32   32   32   32   32   32
 32   32   32   32   32   32
 32   32   32
 32   32

BUFFER DOPO UN GET

 16   8   1   0   143   32
 80   82   71   73   78   77
 69   77   0   54   8   5
 0   159   52   44   52   58
 157   52   58   153   34   80
 82   79   71   82   65   77
 77   65   32   73   78   32
 77   69   77   79   82   73
 65   34   58   153   0   76
 8   10   0   88   178   50
 53   54   172   194   40   52
 52   41   170   194   40   52
 51   41   0   98   8   15
 0   89   49   178   194   40
```

```
88    41    58    89    50    178
194    40    88    170    49    41
0    129    8    17    0    139
89    49    170    89    50    178
48    167    153    88    59    34
42    42    34    59    89    49
59    89    50    58    137    54
48    0    165    8    20    0
153    88    59    34    42    42
34    59    89    49    59    89
50    59    34    76    73    78
75    61    34    59    89    50
172    50    53    54    170    89
49    0    189    8    25    0
89    49    178    194    40    88
170    50    41    58    89    50
178    194    40    88    170    51
41    0    221    8    30    0
```

BUFFER DOPO 193 GET

```
3    1    8    82    9    80
82    71    73    78    77    69
77    32    32    32    32    32
32    32    32    32    32    32
32    32    32    32    32    32
32    32    32    32    32    32
32    32    32    32    32    32
32    32    32    32    32    32
32    32
```

Come puoi vedere dai risultati, dopo la OPEN nel buffer si trova 1 nella prima posizione. Segue nei 2 byte successivi l'indirizzo di inizio del programma (8*256+1=2049). Segue nei 2 byte successivi l'indirizzo di fine programma (9*256+82=2376). Segue il nome del file: PRGINMEM.
Il contenuto del buffer dopo il primo GET# ci mostra il programma byte dopo byte, esattamente come si trova in memoria. Però il contenuto del buffer dopo il GET# numero 193 ci mostra il blocco di chiusura del file, abbiamo cioè perso un pezzo del file, a dimostrare che i file di programma non possono essere letti come file ASCII.

Segue il programma FILEPROG2, che ci consente di vedere il contenuto del blocco di testata di un file di programma.

```
10 REM FILEPROG2
15 REM STAMPA TESTATA FILE PROGRAMMA
20 GOSUB100:REM AZZERA BUFFER
25 OPEN4,4:REM APRE STAMPANTE
30 OPEN1,1,0,"PRGINMEM"
35 PRINT#4,"BUFFER DOPO OPEN PRGINMEM"
40 PRINT#4:GOSUB55
45 CLOSE1:CLOSE4:STOP
50 REM UTCASS
55 REM STAMPA BUFFCASS
60 REM INDIRIZZI BUFFER CASSETTA
65 I1=828:I2=1019
70 FORI=I1TOI2STEP6
75 FORK=0TO5
80 PRINT#4,PEEK(I+K);" ";
85 NEXTK
90 PRINT#4:NEXTI
95 RETURN
100 REM AZZERA BUFFCASS
105 I1=828:I2=1019
110 FORI=I1TOI2
115 POKEI,32
120 NEXTI
125 RETURN
```

Noi abbiamo operato così:

- abbiamo salvato su cassetta il programma PRGINMEM nei diversi modi possibili, cioè con SAVE senza sa, o con sa=1, sa=2 e sa=3,

- abbiamo aperto ognuno di questi file come file in lettura e stampato il contenuto del buffer usando il programma FILEPROG2,
- abbiamo constatato che cambia il primo carattere del bloccco di testata.

Segue il programma FILEPROG3.

```
10 REM FILEPROG3
15 REM STAMPA BUFFER DOPO SAVE PROGRAMMA
20 OPEN4,4:REM APRE STAMPANTE
25 PRINT#4,"BUFFER DOPO SAVE"
```

```
30 PRINT#4:GOSUB45
35 CLOSE1:CLOSE4:STOP
40 REM UTCASS
45 REM STAMPA BUFFCASS
50 REM INDIRIZZI BUFFER CASSETTA
55 I1=828:I2=828+17
60 FORI=I1TOI2STEP6
65 FORK=0TO5
70 PRINT#4,PEEK(I+K);" ";
75 NEXTK
80 PRINT#4:NEXTI
85 RETURN
```

Con questo programma abbiamo operato così:

- abbiamo caricato in memoria FILEPROG3,
- abbiamo eseguito in immediato il SAVE del programma usando i diversi valori di sa consentiti,

- dopo ogni operazione di SAVE abbiamo fatto girare il programma ottenendo in stampa i primi 18 caratteri del buffer; anche in questo caso nel blocco di chiusura del file cambia il primo carattere.

3.4 ESEMPIO DI FILE SEQUENZIALE

Riportiamo ora un programma che crea e gestisce un file sequenziale su cassetta.Il nome del programma è PRFL3C e il nome del file gestito è, in questo esempio, FL3C.

Abbiamo deciso di creare un archivio di indirizzi. Il record logico del file deve comprendere i seguenti 6 campi:

COGNOME
NOME
INDIRIZZO
CAP
CITTA'
TELEFONO.

I campi possono essere di lunghezza variabile e, in conseguenza, il record logico risulta di lunghezza variabile. Abbiamo deciso di tagliare ogni campo in modo che risulti al massimo di 79 caratteri. Inoltre abbiamo deciso di tenere i record in ordine alfabetico di cognome e nome, scartando in ingresso i nomi fuori ordine, con opportuna segnalazione, e non accettando cognomi e nomi uguali.

Il numero di record gestibili dipende dalla lunghezza del nastro e dalla memoria disponibile. Devi tener presente che la casa costruttrice raccomanda di non usare cassette superiori ai 30 minuti. Inoltre, nella fase di aggiornamento del file, dato che si dispone di un solo registratore collegato, deve essere caricato in memoria tutto il file, deve essere modificato e poi riscritto tutto. La memoria occupata dipende dalla lunghezza dei campi dei record.

Il programma esegue le seguenti operazioni:

1) Crea il file ex-novo

2) Lista il file sulla stampante

3) Aggiorna il file consentendo di:

- modificare record,
- cancellare record,
- aggiungere record.

Riportiamo nella figura 3.1 lo schema a grandi blocchi del programma.

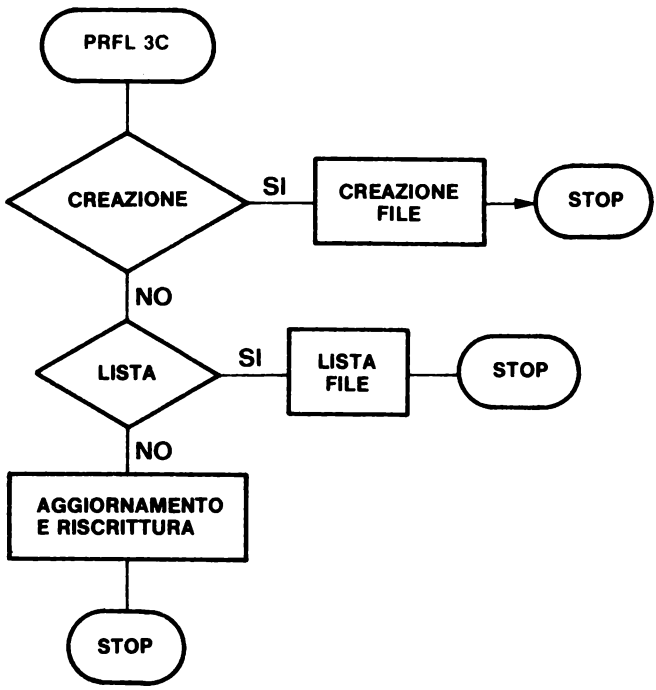


Figura 3.1 Schema a grandi blocchi del programma

Abbiamo costruito il programma usando la tecnica dei sottoprogrammi e in modo che possa essere facilmente modificato per adattarlo alle proprie esigenze.

Il programma non presenta un menù iniziale su tutte le funzioni, ma pone domande successive sulle operazioni possibili. Finita una fase, per passare alle altre si deve ripartire con RUN. Inoltre, se, dopo la creazione del file, si vuole farne la lista o altro, si deve provvedere a riavvolgere il nastro.

Per uscire dalla fase di caricamento dati si deve rispondere con quattro asterischi alla richiesta del COGNOME (****).
Nella fase di aggiornamento viene caricato in memoria tutto il file, dimensionando opportunamente sei vettori di stringhe, dopo aver chiesto il numero dei record che compongono il file. È opportuno, dopo ogni registrazione, segnare sulla cassetta o altrove il numero dei record che sono stati memorizzati. Quando si aggiorna il file è bene conservare la cassetta con la vecchia copia e registrare la nuova su un'altra cassetta.

La fase di aggiornamento si articola dapprima nella correzione dei record esistenti, poi vengono operate eventuali cancellazioni; queste due operazioni possono essere svolte senza rispettare un ordine alfabetico, dato che la ricerca in memoria inizia sempre dalla prima posizione dei vettori. La cancellazione viene operata ponendo tre spazi al posto del cognome; in fase di riscrittura i cognomi che iniziano con spazio vengono saltati.

L'ultima fase dell'aggiornamento è l'aggiunta di nuovi record; in questo caso i nuovi record devono essere forniti in ordine alfabetico di cognome e nome, infatti il programma procede nella scrittura e aggiunge al posto giusto i nuovi record.

Riportiamo nella figura 3.2 lo schema a grandi blocchi della fase di aggiornamento.

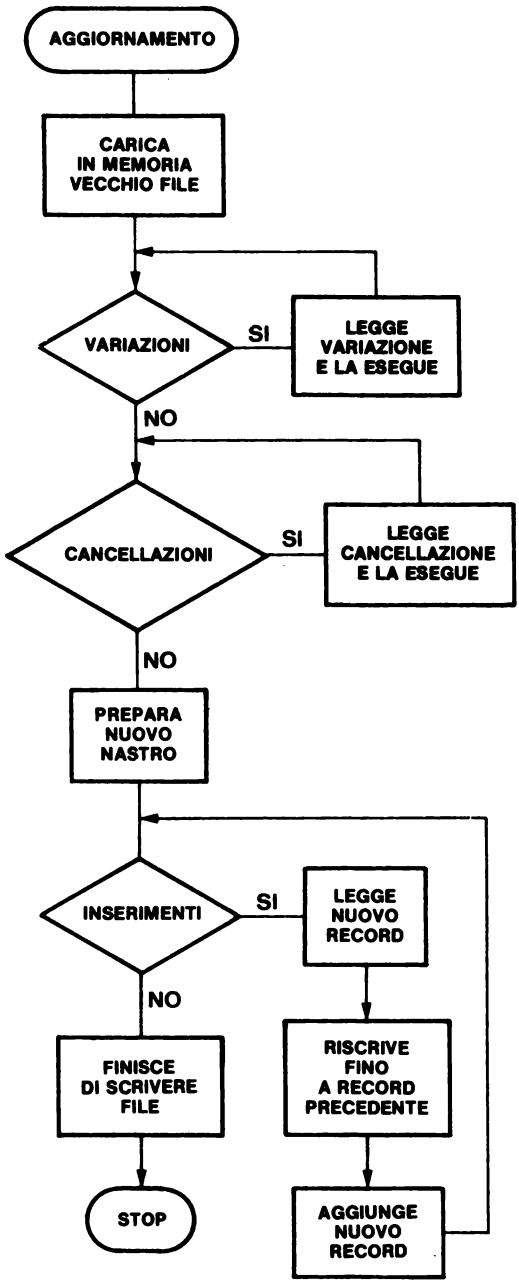


Figura 3.2 Schema a grandi blocchi dell'aggiornamento

Passiamo ad elencare le variabili e le costanti usate nel programma.

PR=4, numero periferica di stampa
NF=4, lfn del file di stampa
D$(6), vettore contenente le descrizioni dei campi dati (COGNOME, NOME, INDIRIZZO, CAP, CITTA', TEL.)
I$(6), vettore contenente i campi dati di un record
CH$=CHR$(13), carattere separatore RETURN
SP$=" " costante contenente 3 caratteri spazio
R$, stringa per le risposte S/N
NF$, nome del file
N, numero massimo dei record in memoria
LC$, vecchio cognome, per controllare l'ordinamento
LN$, vecchio nome, per controllare l'ordinamento
SW, switch di controllo: 1 se finiti dati, 0 altrimenti
N1, puntatore record letti in memoria
C1$(N), vettore dei cognomi in memoria
N1$(N), vettore dei nomi in memoria
I1$(N), vettore degli indirizzi in memoria
P1$(N), vettore dei CAP in memoria
L1$(N), vettore delle CITTA' in memoria
T1$(N), vettore dei telefoni in memoria
FF, switch per gli inserimenti in coda: 1 in coda, 0 no
FS=ST, parola di stato
NR, contatore linee
K, K1, variabili di controllo.

Per passare ad un tipo diverso di record, se rimane di 6 campi, basta modificare le descrizioni alle linee 45 e 50. Se cambia il numero dei campi si devono ricercare nel programma i cicli che fanno riferimento a 6, e modificarli. Risulta più difficile modificare il criterio di ordinamento, che qui agisce sui primi due campi ed opera in senso crescente.

CORRISPONDENZA TRA I BLOCCHI E I NUMERI DI LINEA

1) Creazione del file (linee 55-120)
2) Lista ordinata del file (linee 125-185)
3) Aggiornamento del file (linee 190-400):

    3.1) Modifiche record (linee 255-275)
    3.2) Cancellazioni record (linee 280-310)
    3.3) Riscrittura e inserimenti (linee 315-400).

Segue il listato del programma PRFL3C.

```
10 REM PRFL3C
15 REM CREAZIONE E GESTIONE FILE SEQUENZIALE
20 REM NUMERO DEVICE-NUMERO LOGICO STAMPA
25 PR=4:NF=4
30 REM VETTORI DESCRIZIONI E DATI
35 DIMD$(6),I$(6)
40 CH$=CHR$(13):SP$="    "
45 D$(1)="COGNOME":D$(2)="NOME":D$(3)="INDIRIZZO"
50 D$(4)="CAP":D$(5)="CITTA'":D$(6)="TEL."
55 PRINT"▯▯▯CREAZIONE FILE S/N":INPUT R$
60 IFR$<>"S"THEN125
65 REM***CREAZIONE FILE***
70 GOSUB580:GOSUB595:GOSUB605:GOSUB615
75 LC$="":LN$="":K1=N:FORK=1TON
80 GOSUB405
85 IFSW<>0THENK1=K:K=N:GOTO115
90 IFI$(1)>LC$THEN105
95 IFI$(1)=LC$THENIFI$(2)>LN$THEN105
100 GOSUB685:GOSUB690:GOTO80
105 LC$=I$(1):LN$=I$(2)
110 GOSUB435
115 NEXTK:CLOSE1
120 PRINT"▯FINITO CARICAMENTO   ";K1;" RECORD":STOP
125 PRINT"▯▯▯STAMPA FILE S/N":INPUTR$
130 IFR$<>"S"THEN190
135 REM***STAMPA FILE***
140 GOSUB625:GOSUB580:GOSUB595:GOSUB640:NR=4
145 PRINT#NF,"LISTA FILE ";NF$:PRINT#NF:PRINT#NF
150 GOSUB650
155 PRINT#NF,I$(1);SP$;I$(2)
160 PRINT#NF,I$(3);SP$;I$(4);SP$;I$(5);SP$;I$(6)
165 PRINT#NF:PRINT#NF:NR=NR+4
170 IFFS<>64THEN180
175 CLOSE1:CLOSENF:PRINT"▯▯FINITO LISTA":STOP
180 IFNR=60THENFORK=1TO6:PRINT#NF:NEXTK:NR=0
185 GOTO150
190 REM***AGGIORNAMENTO FILE***
195 PRINT"▯▯▯AGGIORNAMENTO FILE"
200 PRINT"▯▯MONTA NASTRO VECCHIO"
```

```
205 GOSUB690
210 GOSUB595:GOSUB605
215 DIMC1$(N),N1$(N),I1$(N),P1$(N),L1$(N),T1$(N)
220 GOSUB640
225 FORK=1TON
230 INPUT#1,C1$(K),N1$(K)
235 INPUT#1,I1$(K),P1$(K),L1$(K),T1$(K)
240 IFST=64THENCLOSE1:K1=K:K=N:NEXTK:GOTO250
245 NEXTK:PRINT"■FILE SUPERA NUM.MASS.REC.";N:STOP
250 PRINT"■LETTI ";K1;" RECORD":SW=0:N=K1
255 PRINT"■■VARIAZIONI S/N ":INPUTR$
260 IFR$<>"S"THEN280
265 REM***MODIFICA RECORD***
270 GOSUB405:IFSW<>0THEN280
275 GOSUB445:GOTO270
280 PRINT"■CANCELLAZIONI S/N ":INPUTR$
285 IFR$<>"S"THEN320
290 REM***CANCELLAZIONE RECORD***
295 PRINT"RISPONDI **** PER     USCIRE"
300 INPUT"COGNOME ";I$(1):IFI$(1)="****"THEN320
305 INPUT"NOME ";I$(2):GOSUB475
310 GOTO300
315 REM***RISCRITTURA FILE***
320 SW=0:FF=0:K1=0:N1=1
325 PRINT"■PREPARA NUOVO NASTRO":GOSUB580
330 GOSUB595:GOSUB615
335 PRINT"■INSERIMENTI S/N ":INPUTR$
340 IFR$<>"S"THEN390
345 REM***INSERIMENTO NUOVO RECORD***
350 GOSUB405:IFSW<>0THEN385
355 IFFF=1THEN365
360 GOSUB505
365 IFI$(1)>LC$GOTO380
370 IFI$(1)=LC$ANDI$(2)>LN$THEN380
375 GOSUB685:GOTO350
380 GOSUB435:LC$=I$(1):LN$=I$(2):K1=K1+1:GOTO350
385 IFFF=1THEN395
390 GOSUB555
395 PRINT"■FINITO AGGIORNAMENTO"
400 PRINT"SCRITTI ";K1;" RECORD":CLOSE1:STOP
405 REM LETTURA NUOVI DATI
410 SW=0:GOSUB700:IFSW=1THENRETURN
```

```
415 GOSUB720
420 FORJ=1TO6:REM TAGLIA CAMPI TROPPO LUNGHI
425 IFLEN(I$(J))>79THENI$(J)=LEFT$(I$(J),79)
430 RETURN
435 REM SCRIVE NUOVO RECORD SU NASTRO
440 FORJ=1TO6:PRINT#1,I$(J);CH$;:NEXTJ:RETURN
445 REM RICERCA NOME E AGGIORNA
450 FORK=1TON:IFC1$(K)=I$(1)ANDN1$(K)=I$(2)THEN465
455 NEXTK
460 GOSUB490:RETURN
465 I1$(K)=I$(3):L1$(K)=I$(4):P1$(K)=I$(5)
470 T1$(K)=I$(6):K=N:NEXTK:RETURN
475 FORK=1TON:REM RICERCA NOME
480 IFC1$(K)=I$(1)ANDN1$(K)=I$(2)THEN500
485 NEXTK
490 PRINT"■NON TROVATO NOME ";I$(1);SP$;I$(2)
495 GOSUB690:RETURN
500 C1$(K)=SP$:K=N:NEXTK:RETURN
505 REM RICERCA POSIZIONE NUOVO RECORD
510 FORK=N1TON:IFC1$(K)=SP$THEN545
515 IFC1$(K)<I$(1)THEN540
520 IFC1$(K)=I$(1)ANDN1$(K)<I$(2)THEN540
525 IFC1$(K)=I$(1)ANDN1$(K)=I$(2)THEN535
530 N1=K:K=N:NEXTK:RETURN
535 PRINT"NOMI UGUALI":GOSUB690:GOTO530
540 GOSUB660:K1=K1+1:LC$=C1$(K):LN$=N1$(K)
545 NEXTK
550 FF=1:RETURN
555 REM TERMINA SCRITTURA FILE
560 IFN1=NANDFF=1THENRETURN
565 FORK=N1TON:IFC1$(K)=SP$THEN575
570 GOSUB660:K1=K1+1
575 NEXTK:RETURN
580 REM RICHIESTA PREPARAZIONE NASTRO
585 PRINT"■MONTA NASTRO":GOSUB690
590 RETURN
595 REM RICHIESTA NOME FILE
600 INPUT"■NOME FILE ";NF$:RETURN
605 REM RICHIESTA NUMERO RECORD
610 INPUT"■QUANTI RECORD ";N:RETURN
615 REM APERTURA FILE PER SCRIVERE
620 OPEN1,1,2,NF$:RETURN
```

```
625 REM PREPARAZIONE STAMPANTE
630 PRINT"■ACCENDI LA STAMPANTE":GOSUB690
635 OPENNF,PR:RETURN
640 REM APERTURA FILE PER LEGGERE
645 OPEN1,1,0,NF$:RETURN
650 REM LETTURA RECORD DA NASTRO
655 FORJ=1TO6:INPUT#1,I$(J):NEXTJ:FS=ST:RETURN
660 REM SCRITTURA RECORD NASTRO DA VETTORI
665 PRINT#1,C1$(K);CH$;N1$(K);CH$;I1$(K);CH$;
670 PRINT#1,P1$(K);CH$;L1$(K);CH$;T1$(K)
675 RETURN
680 REM MESSAGGIO FUORI ORDINE
685 PRINT"■FUORI ORDINE "I$(1)SP$I$(2):RETURN
690 A$="":GETA$:IFA$=""THEN690:REM ATTESA TASTO
695 RETURN
700 REM INGRESSO NUOVI DATI
705 PRINT"■";:J=1:GOSUB750
710 IFI$(1)="****"THENSW=1:RETURN
715 FORJ=2TO6:GOSUB750:NEXTJ:RETURN
720 REM CONFERMA DATI E CORREZIONE
725 PRINT"■CONFERMI S/N":INPUTR$
730 IFR$="S"THENRETURN
735 INPUT"QUALE CAMPO ";J
736 IFJ<0ORJ>6THEN735
740 PRINTD$(J)" ";:I$(J)=" ":INPUTI$(J)
745 GOSUB760:GOTO720
750 I$(J)=" ":REM RICHIESTA CAMPO
755 PRINTJ;" ";D$(J);" ";:INPUTI$(J):RETURN
760 REM STAMPA DATI SUL VIDEO
765 PRINT"■";:FORJ=1TO6
770 PRINTJ;" ";D$(J);" ";I$(J):NEXTJ:RETURN
775 END
```

NOTA: non devi rispondere mai solo con RETURN alla richiesta di un dato; al limite rispondi con SHIFT-SPAZIO e RETURN.

Dalla linea 405 alla linea 775 sono contenuti i sottoprogrammi. Quando compare un messaggio di segnalazione di una situazione, per proseguire si deve premere un tasto.

Riportiamo un pezzo di listato del file FL3C generato. La stampa è organizzata su due righe per ogni record, pensando che i singoli campi siano ragionevolmente corti, data la loro natura.

```
LISTA FILE FL3C

ARBIATI   LINO
PIAZZA DUOMO 3   22222   MILANO   78654

ALLEGRI   MARIELLA
VIALE ROSE 45   22224   CAGLIARI   678954

ARCIMBOLDI   PIERINO
PIAZZA SIENA 45   45321   MILANO   906543

ARROTINO   GIGI
VIA ROSSA 45   67543   COMO   543216

ASSOLO   MARIA
CORSO VENEZIA 1   45321   VERONA   895432

BELLINI   GIUSEPPE
PIAZZA VERDE 888   65789   BRESCIA   90213

BOLLARI   MARGHERITA
BORGO VECCHIO 90   34521   SIENA   89564
```

Riportiamo un breve commento per i sottoprogrammi.

Linee 405-430 *lettura nuovi dati

- pone SW=0 e usa il sottoprogramma in 700 per leggere il nuovo record,
- se torna con SW=1 esce perchè sono finiti i dati in ingresso,
- se no chiama il sottoprogramma in 720 per la conferma con eventuale correzione dei dati letti,
- taglia i campi troppo lunghi a 79 caratteri e esce.

Linee 435-440 *scrive nuovo record su nastro

- scrive i 6 campi separati da CHR$(13) sul nastro.

Linee 445-470 *ricerca nome e aggiorna

- ricerca cognome e nome tra i record presenti nei vettori in memoria,
- se non li trova chiama il sottoprogramma in 490 per segnalare il fatto e esce,
- se li trova sostituisce i nuovi dati a quelli vecchi e esce.

Linee 475-500 *ricerca cognome e nome

- ricerca cognome e nome in memoria,
- se li trova pone 3 spazi al posto del cognome per eliminare il record e esce,
- se non li trova stampa il messaggio di segnalazione, chiama il sottoprogramma in 690 e poi esce. Qui, linea 490, si trova un'entrata indipendente per l'ultimo pezzo del sottoprogramma.

Linee 505-550 *ricerca posizione nuovo record

- salta nella ricerca i record cancellati,
- va a riscrivere i record, che precedono per cognome e nome il nuovo record, con il sottoprogramma in 660,
- se trova cognome e nome uguali, lo segnala e esce,
- se trova la posizione esce con FF=0,
- se termina i dati memorizzati esce con FF=1.

Linee 555-575 *termina scrittura file

- se il file non è ancora terminato lo finisce di scrivere usando il sottoprogramma in 660.

Linee 580-590 *richiesta preparazione nastro

- chiede di montare il nastro e attende la pressione di un tasto per continuare.

Linee 595-600 *richiesta nome file

- richiede il nome del file da elaborare.

Linee 605-610 *richiesta numero record
- richiede il numero N dei record.

Linee 615-620 *apertura file per scrivere
- apre il file per scrivere.

Linee 625-635 *apertura stampante
- apre la stampante.

Linee 640-645 *apertura file per leggere
- apre il file per leggere.

Linee 650-655 *lettura record da nastro
- legge un record dal nastro.

Linee 660-675 *scrittura record su nastro
- scrive un record sul nastro prelevandolo dai vettori in memoria.

Linee 680-685 *messaggio dati fuori ordine
- segnala il cognome e il nome trovati fuori ordine.

Linee 690-695 *attesa tasto per continuare
- attende la pressione di un tasto per proseguire.

Linee 700-715 *lettura dati da tastiera
- chiede il primo campo (sottoprogramma in 750) e se riceve **** pone SW=1 e esce,
- se no chiede gli altri 5 campi e esce.

Linee 720-745 *conferma dati e correzione
- chiede conferma dei dati presenti sul video,
- se non riceve conferma chiede quale campo modificare e lo modifica,
- se riceve conferma esce.

Linee 750-755 *richiesta campo
- chiede un campo, lo legge e esce.

Linee 760-770 *stampa record sul video
- stampa il record sul video e esce.

5

INDICE

Capitolo 1 - File su disco
1.1 Unità 1541 1
1.2 Dischetto 2
1.3 Disk Operating System (DOS) 9
1.4 Comandi per la gestione del disco 10
1.5 File di programmi 16
1.6 Il file con indice 17

Capitolo 2 - File SEQUENZIALI
2.1 File SEQUENZIALI di dati 19
2.2 Esempio di archivio SEQUENZIALE 28

Capitolo 3 - File RANDOM
3.1 File RANDOM di dati 37
3.2 Esempio di archivio RANDOM 55
3.3 Esempio di archivio RANDOM/USER 70

Capitolo 4 - File RELATIVI
4.1 File RELATIVI di dati 87
4.2 Esempio di archivio RELATIVO 97

Capitolo 5 - Errori
5.1 Messaggi di errore del DOS 107

Capitolo 6 - Programmi di utilità
6.1 Programmi di utilità 111

APPENDICE A — MAGIC DESK I 127
APPENDICE B — CALC RESULT 131
APPENDICE C — Le basi di dati 133

CAPITOLO 1

FILE SU DISCO

1.1 UNITA' 1541

L'unità 1541 è una periferica intelligente; essa contiene il microprocessore 6502, che lavora in modo indipendente dopo aver ricevuto i comandi dalla CPU del calcolatore. Le parti componenti l'unità sono:

- il microprocessore 6502,
- 16K di memoria ROM, contenenti il DOS (Disk Operating System),
- 2K di memoria RAM per i buffer e le memorie di lavoro,
- l'interfaccia seriale tipo IEEE488,
- due porte di comunicazione,
- le parti elettromeccaniche necessarie.

Le due porte di comunicazione consentono di collegare più di una unità 1541 al calcolatore, si può arrivare a cinque unità disco più una stampante, che comunicano a turno tramite il bus seriale.

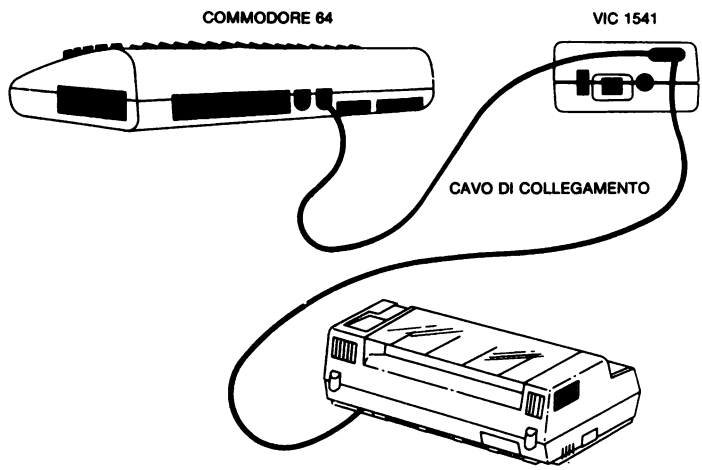


Fig. 1.1 Collegamento tra calcolatore, unità 1541 e stampante

Nella parte anteriore dell'unità sono presenti due indicatori luminosi; il più esterno, a luce verde, se acceso indica che l'unità è pronta a lavorare (READY), il più interno, a luce rossa, sta acceso mentre è in corso fisicamente una operazione di lettura o scrittura da o su floppy. Questo indicatore lampeggia quando si verifica una condizione di errore.

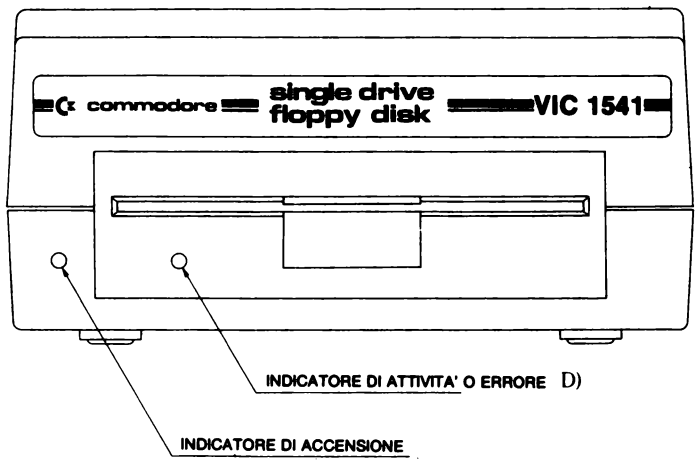


Fig. 1.2 Parte anteriore unità 1541

Al COMMODORE 64 possono essere collegate fino a 5 unità 1541, in tale caso i numeri identificativi partono da 8 e possono arrivare a 12.
I dischetti (floppy) per l'unità 1541 sono gli standard 5 e 1/4", singola faccia, singola densità.

1.2 DISCHETTO

Il dischetto (floppy) è contenuto in una busta protettiva di plastica e deve essere maneggiato con cura.

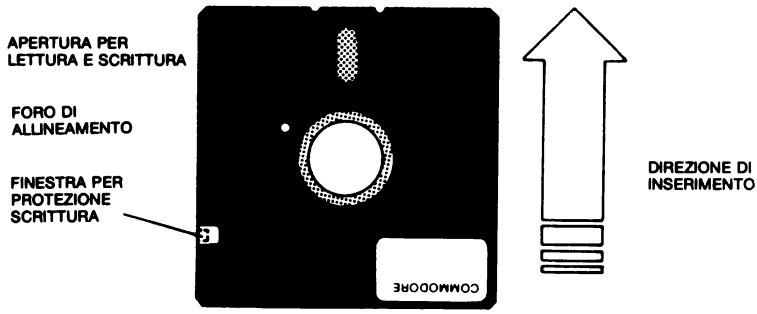


Fig. 1.3 Dischetto

Nella figura 1.3 sono visibili le aperture che reca la busta di plastica protettiva; la più grande serve per comunicare con la testina di lettura e scrittura, il foro piccolo serve per l'allineamento (corretta inserzione), la finestra laterale, se chiusa impedisce di scrivere sul dischetto, e, quindi, serve di protezione. Il dischetto non deve mai essere toccato nelle zone delle aperture. Esso va inserito nell'unità secondo la direzione indicata in figura, quando sulla stessa è acceso l'indicatore luminoso verde. Inoltre, il dischetto non deve mai essere estratto quando è acceso l'indicatore rosso, e non deve essere lasciato in loco quando si toglie corrente all'unità 1541.

Il dischetto viene registrato secondo tracce concentriche; sul tipo usato per l'unità 1541 sono disponibili 40 tracce su una sola faccia, ma ne vengono usate dall'unità solo 35. Su ogni traccia l'unità di informazione trasferibile in una operazione di Input o Output è il SETTORE, che è il blocco fisico di questo tipo di supporto. Il settore è una parte di traccia, delimitata da due raggi che partono dal centro.

Ovviamente le dimensioni fisiche di un settore variano tra tracce diverse, ma la quantità di informazioni registrabile è sempre la stessa, per l'utente sono disponibili 256 byte per settore. Inoltre, il numero di settori disponibili sulle diverse tracce non è sempre lo stesso.

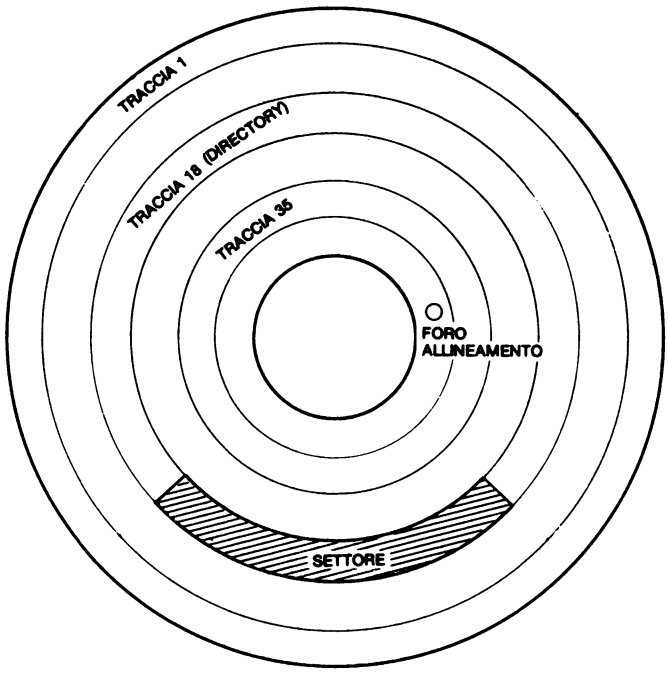


Fig. 1.4 Schema non in scala di un dischetto

Il dischetto quando è nuovo non risulta immediatamente utilizzabile; infatti su di esso non sono state registrate le suddivisioni necesssarie per poterlo usare, cioè le tracce e i settori. Il DOS fornisce il comando necessario per preparare il dischetto all'uso, tale operazione prende il nome di FORMATTAZIONE. Dopo la formattazione il dischetto reca su di sè tre tipi di indicazioni:

- quelle che permettono di distinguerlo da un altro dischetto,
- quelle che danno notizie del suo contenuto,
- quelle che marcano gli indirizzi di traccia e settore.

Il nome che identifica il dischetto è formato da due parti:

- il nome vero e proprio di 18 byte al massimo,
- la ID, identificazione del dischetto di 2 caratteri. Questa indicazione è, come vedremo, molto importante; tra l'altro consente di distinguere tra loro una serie di dischetti ai quali è stato attribuito lo stesso nome.

Nella tabella 1.1, che segue, riportiamo la suddivisione in tracce e settori.

Tab. 1.1 FORMATO DEL DISCHETTO

Numero traccia	Numero settori	Numerazione settori
da 1 a 17	21	da 0 a 20
da 18 a 24	19	da 0 a 18
da 25 a 30	18	da 0 a 17
da 31 a 35	17	da 0 a 16

Come risulta dalla tabella 1.1, sono disponibili:

```
17 tracce da 21 settori : 17*21 = 357
 7 tracce da 19 settori :  7*19 = 133
 6 tracce da 18 settori :  6*18 = 108
 5 tracce da 17 settori :  5*17 =  85
----------------------------
totale settori                  = 683
```

La traccia 18, di 19 settori, ha un utilizzo particolare, che passiamo a descrivere. Il dischetto non può essere utilizzato se non contiene un indice di riferimento, come un qualunque libro, dove siano registrati i nomi dei file e i loro indirizzi sul dischetto stesso. Infatti il dischetto non è come il nastro un supporto di tipo sequenziale, su di esso si può registrare dove si vuole, e, se non si sa dove sono registrati i file, non si può utilizzarli.

La traccia 18 è utilizzata nel seguente modo:

• settore 0, byte da 0 a 143,per la mappa di occupazione del dischetto, chiamata BAM (Block Availability Map);

• settore 0, byte da 144 a 255, per l'identificazione del dischetto (blocco di testata per l'indice del disco, che viene chiamato DIRECTORY);

• settori da 1 a 19 indice del disco (DIRECTORY).

I 144 byte della BAM sono utilizzati come segue:

• byte 0 e 1, come concatenamento al settore seguente, contengono 18 e 01 (traccia 18, settore 1);

• byte 2, contiene 65, codice ASCII della lettera A, che indica il formato dell'unità, tipo 4040;

• byte 3, contiene lo 0 binario (tutti bit 0) e non ha significato,

• byte da 4 a 143, contengono la mappa dei settori occupati: sono utlizzati 4 byte per ogni traccia, quindi si hanno 35 gruppi di 4 byte (35*4=140). Ogni quartina è utilizzata così:

• primo byte: numero settori ancora disponibili nella traccia,

• secondo byte: mappa dei primi 8 settori della traccia, utilizzando i bit a partire dai meno significativi (il bit più a destra per il settore 0, il bit più a sinistra per il settore 7),

• terzo byte: mappa degli altri 8 settori della traccia dal settore 8 (bit più a destra) al settore 15 (bit più a sinistra),

• quarto byte: mappa degli altri settori della traccia a partire dal 16. Per il gruppo di tracce con 21 settori sono utilizzati gli ultimi 5 bit a destra, per il gruppo di tracce con 19 settori sono utilizzati gli ultimi 3 bit, per quello con 18 settori gli ultimi 2 e per quello con 17 settori solo l'ultimo bit.

Come vedi, la mappa del disco si serve di un bit per ogni settore, se il bit è 1, il settore è libero, se è 0, il settore è occupato.

I 112 byte della seconda parte del settore 0 sono utilizzati per l'identificazione del dischetto in questo modo:

• byte da 144 a 161, 18 byte utilizzati per il nome del disco, se il nome è più corto ad esso viene aggiunto il carattere di codice ASCII 160, corrispondente a spazio + SHIFT (spazio inverso);

• byte 162 e 163, i due caratteri della ID del disco;

• byte 164, un codice 160;

• byte 165 e 166, contengono i caratteri 2A per versione del DOS e formato dischetto;

• byte da 167 a 170, 4 codici 160;

• byte da 171 a 255, sono 85 byte non utilizzati.

I settori da 1 a 18 della traccia 18 sono utilizzati per la DIRECTORY del disco; in ogni settore è possibile registrare l'indice di 8 file. In conseguenza un disco può contenere solo 144 file diversi (18*8=144), e, se ciascuno di essi è abbastanza corto, possono restare settori inutilizzati. Come vedremo nel seguito, è anche possibile registrare settori del disco senza mantenerne indicazione nell'indice, però è una pratica non raccomandabile.

Ogni registrazione di file nell'indice prende il nome di ENTRATA (ENTRY) e occupa 30 byte.

La struttura di ogni settore è la seguente:

- byte 0 e 1, contengono il concatenamento al settore seguente, la traccia in 0 e il settore in 1; se il settore è l'ultimo della catena il byte 0 contiene lo zero binario e il byte 1 contiene il puntatore all'ultimo byte utilizzato (255 se tutto il settore),
- byte da 2 a 31, entrata 1,
- byte 32 e 33, 2 caratteri separatori contenenti zero binario,
- byte da 34 a 63, entrata 2,

....

....

- byte 224 e 225, 2 caratteri separatori,
- byte da 226 a 255, entrata 8.

Per ogni entrata di file sono registrate le seguenti informazioni:

- byte 0, tipo del file:

tutti bit 0 per file cancellato (DEL),
128 + 1 per sequenziale (SEQ)
128 + 2 per programma (PRG)
128 + 3 per utente (user) (USR)
128 + 4 per relativo (REL),

- byte 1 e 2, indirizzo di traccia e settore del primo blocco del file,
- byte da 3 a 18, 16 byte che contengono il nome del file, se il nome è più corto sono aggiunti codici 160,
- byte 19 e 20, usati solo per i file relativi, contengono gli indirizzi di traccia e settore del primo blocco SIDE SECTOR (indice interno del file relativo),
- byte 21, usato solo per i file relativi, contiene la lunghezza del record, minore di 255,
- byte da 22 a 25, 4 byte non usati,
- byte 26 e 27, traccia e setttore del file rimemorizzato, cioè del file che è stato memorizzato con lo stesso nome dopo una OPEN@,
- byte 28 e 29, numero di blocchi occupati dal file, precede il byte basso (nel calcolo: contenuto byte 28 + 256 * contenuto byte 29).

Precedentemente abbiamo calcolato che nel dischetto sono disponibili 683 settori, dato che la traccia 18 è usata per la DIRECTORY, i settori disponibili per l'utente sono 683-19=664. Se, dopo la formattazione del dischetto, richiedi la stampa della directory, puoi vedere le indicazioni sulla identificazione del dischetto, nessuna entrata di file, e ti vengono dichiarati liberi 664 blocchi.

Quando inserisci il dischetto nell'unità 1541 vedi accendersi per poco la spia rossa e senti girare il dischetto. Avviene l'operazione di INIZIALIZZAZIONE, da non confondere con la formattazione; essa consiste nel fatto che il dischetto viene fatto girare fino ad ottenere l'allineamento della testina di lettura con l'inizio delle tracce e viene letto il settore 0 della traccia 18 in uno dei buffer di 256 byte dell'unità. Questa operazione è ESSENZIALE, infatti, se non è nota la situazione di occupazione delle tracce e dei settori, si rischia di andare a scrivere su settori già precedentemente occupati. La BAM deve risiedere nella RAM dell'unità 1541 per poter operare sul dischetto e deve essere riscritta sul dischetto quando viene modificata la situazione. Per questa ragione è assolutamente necessario chiudere i file sui quali si opera; l'operazione di chiusura fa avvenire, se necessario, la riscrittura della BAM. Un dischetto, che contenga una BAM non corrispondente alla sua reale situazione di occupazione dei settori, è praticamente inutilizzabili con le normali operazioni BASIC.

Riepiloghiamo la differenza che esiste tra le due operazioni: FORMATTAZIONE e INIZIALIZZAZIONE:

- la formattazione agisce sul supporto dischetto, annulla le precedenti registrazioni e memorizza la nuova identificazione, dopo aver registrato gli indirizzi di traccia e settore;
- l'inizializzazione allinea il dischetto nell'unità e carica nella RAM dell'unità le informazioni necessarie.

Ovviamente non viene caricata la DIRECTORY; ci vorrebbe troppo spazio. La DIRECTORY viene letta quando si apre un file; essa serve per fornire l'indirizzo di inizio del file e le altre informazioni necessarie per il tipo di file in uso. Essa viene in parte riscritta quando si chiude un file ed è necessario registrare modifiche nell'entrata ad esso relativa.

Abbiamo parlato di settori (BLOCCHI FISICI) di 256 byte. In realtà ogni settore contiene più di 256 byte, infatti in esso sono registrati anche alcuni caratteri di controllo, gli indirizzi di traccia e settore, un numero che rappresenta (CHECKSUM) la somma di tutti i bit 1 registrati nella parte di settore a disposizione dell'utente, e serve a controllare la bontà della registrazione. In sostanza in ogni settore sono presenti due parti: quella a disposizione del sistema, con un suo controllo di checksum, e quella a disposizione dell'utente di 256 byte.

Al programmatore interessa imparare a gestire i suoi 256 byte. Questi, in realtà, si riducono a 254, infatti il byte 0 e il byte 1 sono usati dal sistema per concatenare tra loro i settori di un file, ponendo in essi l'indirizzo della traccia e del settore logicamente seguente, che di solito non è quello fisicamente adiacente. Approfondiremo meglio queste cose nel seguito.
Tra i settori esiste una zona disco inutilizzata, chiamata GAP.

Riportiamo nella tabella 1.2 le caratteristiche del dischetto.

Tab. 1.2 CARATTERISTICHE TECNICHE DISCHETTO

Capacità totale: 174848 byte (683*256)
Capacità per file sequenziali: 168656 byte (664*256-664*2 (byte di concatenamento))
Capacità per file relativi: 167132 byte (664*256 - 664*2 (concatenamento) - 254*6 (side sector)), con numero massimo di record logici per file 65535
Entrate nella directory: 144
Settori per traccia: da 17 a 21
Byte per settore: 256
Tracce: 35
Totale settori: 683, di cui disponibili 664

1.3 DISK OPERATING SYSTEM (DOS)

Il DOS (Disk Operating System) risiede nella ROM del disco; esso provvede all'esecuzione dei comandi che vengono trasmessi all'unità 1541 dal Sistema Operativo del calcolatore. Il calcolatore invia comandi o dati tramite l'interfaccia seriale, il DOS interpreta i comandi e li esegue, scrive o legge il dischetto, gestisce la BAM e la DIRECTORY.

Il linguaggio BASIC mette a disposizione i 5 comandi già noti:

OPEN, CLOSE, PRINT#, GET#, INPUT#

- il parametro "lfn", numero logico del file, ha sempre il solito significato di numero logico identificatore del file,
- il parametro "dn", se si usa una sola unità 1541, ha il valore 8,

- il parametro "sa", noto fino ad ora come "indirizzo secondario", gioca un ruolo molto importante, infatti in base al suo valore il DOS stabilisce se riceve comandi o dati dal calcolatore.

Seeondo la terminologia in uso nei manuali della Commodore, il parametro "sa" viene chiamato CANALE. Nel seguito ci atteniamo anche noi a questa terminologia. Il CANALE può avere valori da 0 a 15, con i seguenti significati:

- sa=15, apre il canale 15 per invio di comandi al DOS, o per invio di segnalazioni speciali da parte del DOS;
- sa=0 e sa=1, aprono rispettivamente i canali 0 e 1 per l'esecuzione dei comandi LOAD e SAVE, cioè trasferimento di programmi tra disco e calcolatore e viceversa;
- sa=2,...,14, apre il corrispondente canale per il trasferimento di dati.

Sia con il comando OPEN, che con il comando PRINT#, possono essere inviate al DOS stringhe di comandi, come vedremo nel seguito. Quando il DOS riceve un comando che comporta l'apertura di un canale, esso assegna al canale lo spazio necessario per lavorare e mette a disposizione uno dei suoi buffer in RAM.
Sull'unità sono disponibili 8 buffer di 256 byte ciascuno; di essi 4 sono impegnati per il canale dei comandi, la BAM, le variabili di lavoro e il controllo delle operazioni. Restano, in conseguenza, liberi solo 4 buffer e il numero dei file gestibili contemporaneamente risulta di 3 o di 4, a seconda del tipo di file.
Il DOS consente di gestire 4 tipi di file:

- Programmi (PRG)
- Sequenziali (SEQ)
- Random (USR, o senza specifica di tipo, cioè collegabili con gli User, che sono sequenziali, per farne comparire l'entrata nella Directory)
- Relativi (REL).

1.4 COMANDI PER LA GESTIONE DEL DISCO

Ci occupiamo qui dei comandi che consentono di operare con il dischetto, indipendentemente dal tipo di file coinvolto; essi possono essere usati sia in modo immediato, che da programma. Questi comandi sono già stati trattati nel volume dedicato al BASIC, ma ci sembra utile riportarli anche in questa sede per completezza nell'esposizione dell'argomento.
Si tratta di una serie di comandi da inviare al disco sul canale 15 dei comandi. L'operazione necessaria per creare la comunicazione è:

OPEN lfn,8,15

cioè abbiamo usato: dn=8 e sa=15.
Nel seguito usiamo sempre dn=8, dato che supponiamo di avere una sola unità 1541 collegata al calcolatore.
Abbiamo usato sa=15 perchè dobbiamo aprire il canale dei comandi.
Di solito, solo per la comodità di ricordare un solo numero, noi usiamo:

OPEN 15,8,15

cioè poniamo lfn=15, ma lfn può essere un numero qualunque, purchè minore di 255, anzi meglio se minore di 127. Abitualmente per lfn si usano numeri piccoli.
A questa OPEN, che apre il canale dei comandi, può seguire una istruzione:

PRINT# lfn, stringa-comandi

che trasmette il comando al DOS. Si può, però, procedere in altro modo, incorporando la stringa dei comandi nella OPEN stessa, così:

OPEN 15,8,15,stringa-comandi

ottenendo esattamente lo stesso effetto. In ambedue i casi l'operazione deve essere conclusa con:

CLOSE 15 (o CLOSE lfn)

infatti se non adoperi la CLOSE, e in seguito usi ancora OPEN con lo stesso numero per lfn, avrai una segnalazione di errore, perchè cerchi di aprire un canale già aperto.

Dopo queste premesse, nel seguito esaminiamo le stringhe comando da trasmettere in uno dei due modi possibili. Devi tener presente che una stringa comando non deve superare 40 caratteri.

I comandi disponibili sono:

NEW, per la formattazione del dischetto
INITIALIZE, per l'inizializzazione del dischetto
VALIDATE, per sistemare la BAM in base alle entrate della Directory
COPY, per copiare un file
RENAME, per cambiare nome a un file
SCRATCH, per cancellare un file.

Ogni comando può essere citato usando il nome per esteso, o limitandosi alla prima lettera del nome.

NEW

può servire per due scopi diversi:

- preparazione di un disco nuovo per l'uso: vengono registrati gli indirizzi di traccia e settore, viene registrato il nome e la identificazione del disco, viene preparata la BAM e viene predisposto lo spazio per la Directory;
- cancellazione di un disco già usato, mantenendo la stessa identificazione, ma senza riscrivere gli indirizzi.

La stringa-comando si scrive:

"Ndr:fn,xx"

dove:

N identifica il comando e può essere anche NEW,
dr, è il numero del drive, e, se è collegato una unità con un solo disco, come la 1541, risulta 0 e può essere omesso (l'unità 4040 ha due drive, per esempio, 0 e 1),
fn, è il nome del disco,
xx, è la ID (identificazione del disco).

Per formattare un dischetto assegnandogli il nome MAGAZZINO e la ID=33 si può scrivere:

OPEN15,8,15,"N0:MAGAZZINO,33":CLOSE15

oppure:

OPEN15,8,15:PRINT#15,"N0:MAGAZZINO,33":CLOSE15

Qualora nel comando si omettano i due caratteri della ID e si usi un disco già precedentemente formattato, si ottiene la cancellazione del contenuto del dischetto, che mantiene la ID già registrata; se il nome è diverso dal precedente, esso viene modificato. Ovviamente tale operazione, diciamo ridotta, risulta più veloce di una formattazione completa. NON SI PUO' USARE IL COMANDO SENZA ID PER FORMATTARE UN DISCO NUOVO.

INIZIALIZE

serve per allineare la testina di lettura e scrittura all'inizio della traccia e per caricare la BAM nella memoria dell'unità. Meglio eseguire troppo spesso questa operazione, che non eseguirla mai. In realtà, quando si inserisce il dischetto nell'unità l'operazione di inizializzazione deve avvenire automaticamente, ma, in qualche caso, si può avere qualche intoppo. Ti raccomandiamo quindi di eseguire da

programma, o in immediato, l'inizializzazione, quando cambi il dischetto. Non eseguire da programma la funzione "I", se non hai prima chiuso i file. La stringa è molto semplice:

"I" oppure "I0" oppure "INITIALIZE"

VALIDATE

serve per rimettere ordine in un dischetto disastrato; si deve usare il comando se è successo qualcosa di irregolare, e se, per esempio, ti accorgi che la somma dei blocchi occupati più quelli liberi (basta listare la Directory) non dà 664. Una cosa irregolare può essere l'interruzione di un programma, prima che siano andate a buon fine tutte le operazioni di scrittura dei file su disco, cioè senza chiudere i file aperti (in questo caso compaiono degli asterischi nella Directory). Il commando agisce in questo modo:

- rigenera la BAM in base alle entrate regolari presenti nella Directory,
- cancella dalla Directory le entrate irregolari.

Abbiamo visto che nell'entrata di un file è registrato l'indirizzo di traccia e settore del primo blocco; i blocchi successivi sono concatenati tra loro per mezzo dei primi due byte di ogni settore, e il blocco di chiusura reca nel primo byte lo zero binario. Il sistema, seguendo la catena, può rigenerare lo stato della BAM. Da quanto abbiamo detto risulta che NON DEVE ASSOLUTAMENTE essere eseguita questa operazione se un dischetto contiene delle registrazioni, tipo file RANDOM, che non hanno una corrispondente adeguata entrata nella Directory. La stringa è molto semplice:

"V0" oppure "V" oppure "VALIDATE"

COPY

consente di ottenere copie di un file assegnandogli nomi diversi, oppure di fondere più file, fino a 4, in un unico file. Chiamiamo "dfn" il nome del file destinazione, e "sfn" il nome del file sorgente. Il file sorgente (o i file sorgenti) deve essere stato regolarmente chiuso prima di questa operazione. Le stringhe comando possibili sono:

"C0:dfn=sfn" per una copia

"C0:dfn=sfn1,sfn2,sfn3,sfn4" per fondere più file

ovviamente il file sorgente resta inalterato.

Se i FILE sorgente sono più di uno si possono copiare solo FILE sequenziali di dati, no programmi.

RENAME

serve per cambiare il nome di un file. L'operazione differisce dalla copiatura, infatti il file resta registrato dove si trova, ma, nella sua entrata nella Directory, viene cambiato il nome. Chiamiamo "nfn" il nome nuovo, e "ofn" il nome vecchio. Si scrive:

"R0:nfn=ofn" oppure "RENAME:nfn=ofn"

Il file ofn deve essere stato correttamente chiuso prima di operare con RENAME.

SCRATCH

cancella i file che non servono più dal disco, ponendo il tipo DEL, cioè tutti bit 0, nel primo byte della sua entrata nella Directory, e rendendo disponibili nella BAM i settori che il file occupava. Con un solo comando possono essere cancellati più file. Si scrive:

"S0:fn" per cancellare un file

"S0:fn1,fn2,...,fnl" per cancellare l file.

Ti consigliamo di usare il comando SCRATCH quando desideri riscrivere un file che esisteva precedentemente, invece di usare il carattere "@" nella OPEN, o nella SAVE, per i file di programma. Infatti il carattere "@" produce a volte risultati non desiderabili.

Per caricare in memoria la Directory del dischetto si usa questa sequenza:

OPEN15,8,15:LOAD"$",8

se vuoi listarla sul video, devi scrivere:

LIST

se, invece, vuoi listarla sulla stampante:

OPEN4,4:CMD4:LIST

e poi: PRINT#4:CLOSE4

Il sistema usa anche per i file su disco la parola di stato ST; quando ST=64, dopo una operazione di lettura, significa che si è raggiunta la segnalazione di fine file.

Il DOS consente di fare una analisi più precisa dell'andamento delle operazioni disco, richiamando tramite il canale 15 (comandi/errori) quattro variabili, che di solito si usa chiamare, con significato mnemonico: EN, EM$, ET, ES. Il significato del contenuto delle variabili è il seguente:

EN, numero del messaggio di segnalazione,
EM$, descrizione del messaggio,
ET, numero della traccia del dischetto che ha generato la segnalazione,
ES, numero del settore nella traccia.

Al posto di queste variabili, puoi usarne anche altre, esse possono essere tutte variabili stringa, oppure, come abbiamo fatto noi, solo la seconda stringa, e le altre numeriche.

Devi operare così:

OPEN15,8,15:PRINT#15,EN,EM$,ET,ES:CLOSE15

Quando scrivi un programma che gestisce file su disco devi ricordare le seguenti cose:

- Aprire il canale 15, che serve anche per la segnalazione degli errori.
- Non chiudere il canale 15, se nel programma sono ancora aperti altri canali, infatti la chiusura del canale 15 provoca la chiusura di tutti gli altri canali per il DOS, mentre nel programma (tabellina dei file aperti nella RAM del calcolatore) i file risultano ancora logicamente aperti.

- Qualora avvenga una segnalazione di errore, dal punto di vista della logica del programma, vengono chiusi tutti i file, che però restano aperti per il DOS, allora devi scrivere in immediato:
OPEN15,8,15:CLOSE15 per sistemare le cose.

- I comandi NEW, INITIALIZE e VALIDATE chiudono tutti i canali associati al disco e quindi rovinano eventuali file che non siano prima stati chiusi correttamente.

1.5 FILE DI PROGRAMMI

Abbiamo già visto nel secondo Volume come si trattano i file di programmi. Dopo aver studiato, nei precedenti paragrafi, il comportamento dell'unità 1541, appare evidente che un programma non può essere salvato su dischetto o richiamato da esso in memoria, se il dischetto non è stato correttamente inizializzato. Le operazioni di LOAD e SAVE usano implicitamente sa=0 e sa=1.
Devi fare attenzione a quanto segue:

• Se richiami da disco con LOAD un programma che non esiste, comincia a pulsare l'indicatore rosso di errore; in tale caso devi scrivere:

CLOSE15:OPEN15,8,15,"I"

e poi richiamare con LOAD un programma usando il nome giusto.

• Se salvi con SAVE su disco un programma con un nome che esiste già, il salvataggio non ha luogo. Puoi usare SAVE con il carattere "@", ma è meglio seguire la procedura consigliata a proposito del comando SCRATCH.

• Se non sei sicuro che una operazione disco sia andata a buon fine controlla la Directory. Ricordati che caricando in memoria la Directory si cancella il programma eventualmente presente. Questo non succede se inizialmente è stato caricato in memoria e mandato in esecuzione il programma di utilità C-64 WEDGE.

• Dopo il SAVE di un programma esegui sempre il VERIFY.

Nel quarto Volume abbiamo cercato di leggere un file di programmi su cassetta come file sequenziale e siamo andati ad analizzare il contenuto del buffer. Nel caso del dischetto possiamo soddisfare più facilmente la curiosità di vedere come è registrato un programma sul supporto.
Consideriamo sempre il programma PRGINMEM e operiamo come segue.
Carichiamo in memoria il programma DCOMEFS, riportato nel Paragrafo 3.15, e, dopo aver stampato la directory del dischetto che contiene il programma PRGINMEM, rileviamo l'indirizzo del primo blocco del nostro programma. Nel caso del nostro dischetto abbiamo trovato che il primo blocco sta nella traccia 19, settore 7, e che il programma occupa 2 blocchi.
Carichiamo in memoria dal dischetto TEST/DEMO, contenuto nella scatola dell'unita 1541, il programma DISPLAY T&S, montiamo il dischetto contenente il programma PRGINMEM, e facciamo girare il programma DISPLAY T&S. Chiediamo l'uscita su stampante e diamo come indirizzo di traccia e settore: 19, 7.

Otteniamo così il contenuto del primo blocco, poi continuiamo con il secondo blocco e passiamo ad analizzare lo stampato. L'interpretazione non è semplicissima, dato che non possiamo considerare la parte stampata a destra, ma solo i codici numerici riportati a sinistra in esadecimale. Se proviamo a fare qualche conversione troviamo tutti i caratteri del nostro programma, come sono riportati a pag. 267 del primo volume. I primi due byte del primo blocco contengono gli indirizzi di traccia e settore del secondo blocco, sempre in esadecimale, il secondo blocco ha l'indirizzo di traccia a zero binario per rompere la catena, mentre il secondo byte contiene il puntatore al byte di fine file nel blocco. Constatiamo così che il file di programma (tipo PRG) termina con due byte a zero binario, dopo il byte a zero binario che chiude l'ultima istruzione.

1.6 IL FILE CON INDICE

I file sequenziali si elaborano molto semplicemente, ma sono poco flessibili, in quanto si possono solo ricercare i record che interessano partendo dal primo e procedendo in sequenza. Nei nostri esempi di archivio lavoravamo su record organizzati secondo l'ordinamento di due campi. Ovviamente si possono anche avere archivi senza alcun ordinamento, ma in questo caso i tempi di ricerca si allungano, dato che si rischia di dover leggere tutto il file per concludere che il record cercato non esiste.

Nel seguito trattiamo i file RANDOM e RELATIVI, cioè file nei quali si può accedere a un qualunque record, o per mezzo del suo indirizzo fisico, o per mezzo del numero d'ordine del record nel file. Questo fatto rende la gestione molto flessibile, ma sussiste un grosso problema: come si fa a sapere qual è l'indirizzo fisico di un record di un file RANDOM, o, analogamente, come si fa a sapere qual è il numero d'ordine nel file RELATIVO di un record?

La prima risposta che viene in mente è la seguente: si tengono degli elenchi ordinati su carta con queste informazioni e si va a cercare quello che interessa.
La cosa non è molto elettronica!

Precisiamo alcuni concetti. Ogni file viene organizzato di solito in base a una chiave o a un gruppo di chiavi, cioè in base al contenuto di alcuni campi dei suoi record. Quando si crea il file si decide quale è la chiave di ordinamento più importante e si organizzano i record in base a tale chiave. La situazione ideale è che la chiave sia la sequenza dei numeri naturali, che risulta anche un campo significativo del record. In tale caso il campo chiave rappresenterebbe il numero d'ordine del record nel file e sarebbe immediata la ricerca su un file RELATIVO. Per un file RANDOM, non

sarebbe difficile, tenendo conto della lunghezza del record, impostare un algoritmo che consenta di risalire dal numero d'ordine del record al suo indirizzo fisico su disco, conoscendo l'indirizzo del primo blocco.

Sfortunatamente il caso citato non si presenta quasi mai. Per risolvere il problema, quando si crea un file, chiamiamolo PRINCIPALE, si crea contemporaneamente un secondo file, chiamiamolo INDICE, che ha la caratteristica di essere formato da record molto corti, contenenti il campo chiave e l'indirizzo del record corrispondente nel file principale. Il file indice, essendo corto, può essere letto in memoria con poche operazioni di lettura, e, se possibile, mantenuto in memoria durante l'esecuzione dei programmi che trattano il file principale, consentendo rapide ricerche in base alla chiave del record.
Se, per esempio, un file principale ha il record di 256 byte, mentre il campo chiave di ordinamento è di 10 byte, supponendo che per l'indirizzo bastino 3 byte, ne consegue che il file indice può avere il record di 13 byte. In un settore possono essere contenuti 16 record, quindi leggendo un settore del file indice possono essere compiute ricerche su 16 record.
Molti calcolatori, di dimensioni maggiori del nostro, hanno già incorporata nel Sistema Operativo la gestione dei file con indice. Noi possiamo crearcela su misura per le nostre esigenze.
Introdotto il concetto di file con indice, si vede immediatamente come partendo dall'indice, diciamo primario, di un file se ne può creare un altro, usando come chiave un altro campo, per ottenere una particolare elaborazione.

CAPITOLO 2

FILE SEQUENZIALI

2.1 FILE SEQUENZIALI DI DATI

I file SEQUENZIALI di dati su disco hanno caratteristiche analoghe a quelle dei file sequenziali su cassetta.
L'utente deve stabilire la struttura del record logico e dei campi che lo compongono. I campi possono essere di lunghezza fissa o variabile e in conseguenza anche i record logici. L'utente non deve preoccuparsi del collegamento tra record logico e blocco fisico; esso è a carico del sistema. I dati vengono inviati dal calcolatore all'unità 1541 carattere per carattere, il DOS provvede ad ammucchiarli nel buffer messo a disposizione al momento della OPEN del file. Quando il buffer è pieno i dati vengono scritti in un settore del dischetto; i primi due byte del settore sono usati per il concatenamento al settore successivo. In questo caso non è noto a priori l'indirizzo del buffer, che risiede nella RAM dell'unità 1541.
Ogni campo deve essere separato dal campo seguente da un carattere separatore valido, che può essere la virgola (CHR$(44)) o il RETURN (CHR$(13)). Vale la regola che, se i dati devono essere letti con INPUT#, non possono essere presenti più di 79 caratteri tra due RETURN. Inoltre, se si è usato come carattere separatore registrato la virgola, tutti i campi compresi tra due RETURN devono essere letti con una sola istruzione INPUT#, che rechi nella lista un numero sufficiente di nomi di variabili. Qualora il file venga letto solo con GET#, si ha una maggiore libertà. I separatori usati nella lista dati della PRINT# tra le variabili agiscono come per la cassetta: la virgola aggiunge spazi, il punto e virgola no.

Si può interrogare la parola di stato ST per accorgersi della fine del file in lettura (ST=64), oppure si può analizzare, servendosi del canale 15, la situazione delle variabili EN, EM$, ET e ES.
I file devono essere aperti con OPEN, elaborati in scrittura o lettura, e chiusi con CLOSE. Se si opera correttamente viene registrata, nella fase di scrittura del file, una entrata nella Directory, recante tutte le indicazioni necessarie a gestire il file. Come per la cassetta, questi file possono solo essere scritti in sequenza partendo dal primo record, oppure letti in sequenza. L'aggiornamento di un file sequenziale su dischetto può solo essere ottenuto riscrivendo completamente il file. Però, nel caso del dischetto, si può lavorare meglio che con la cassetta, infatti per aggiornare un

file si può:

- aprire il file preesistente in lettura,
- aprire un nuovo file con nome diverso per scrittura,
- copiare ordinatamente i record dal vecchio file nel nuovo file, apportando via via le modifiche o le aggiunte,
- chiudere alla fine i due file, cancellare il vecchio file, e cambiare il nome al nuovo file, attribuendogli il nome vecchio.

Passiamo ora ad esaminare le istruzioni BASIC disponibili.

OPEN lfn,dn,sa,"dr:fn,ft,md"

deve essere usata per aprire un file. I parametri, che possono essere costanti o variabili, hanno il seguente significato:
lfn, numero logico del file, da 1 a 127
dn, 8 per una unità 1541
sa, indirizzo secondario da 2 a 14 (canale)
dr, numero dell'unità, 0 o si omette
fn, nome del file, è quello che compare nella directory
ft, tipo=S
md, modo, può essere: W per scrivere
R per leggere.

Nella stringa, dopo sa, si può inserire prima dei ":" il carattere "@", se md=W, per ottenere di cancellare un eventuale file già presente con lo stesso nome, prima di creare il nuovo file. Ti consigliamo di usare il comando SCRATCH quando vuoi cancellare un file, e di ricorrere raramente al carattere "@".
La OPEN svolge la sua solita azione dal punto di vista del programma BASIC, e, inoltre, provoca da parte del DOS quelle azioni che sono necessarie per poter trattare un file su dischetto. Tali azioni sono diverse in dipendenza dal carattere md. Infatti, se md=R, viene cercato nella Directory il file indicato, e, se non trovato, viene segnalato un errore, mentre se md=W, e viene trovato un file con il nome indicato (in assenza del carattere @ nella stringa) viene segnalato un errore. Viene in tutti i casi predisposto un buffer per le operazioni relative al file.

PRINT#lfn,lista

provoca la scrittura dei dati della lista sul file contraddistinto dal numero logico lfn. Non ripetiamo le considerazioni già fatte sulla "lista".
Ti facciamo, però, notare, che in questo caso possono essere aperti contemporaneamente più file su disco, per lettura e scrittura, e si può operare di volta in volta su quello che interessa e viene specificato dal numero lfn. Il numero massimo di file sequenziali gestibili contemporaneamente sul dischetto è 3.

INPUT#lfn,lista

legge dal file, aperto in lettura con il numero logico lfn, tanti dati quanti corrispondono alle variabili della lista. Se i nomi delle variabili sono di tipo errato (variabili numeriche in presenza di dati non numerici), si ha segnalazione di errore. Analogamente, se i dati contengono troppi caratteri (stringhe più lunghe di 79 caratteri tra due RETURN) si ha una segnalazione di errore. Ricordiamo che si possono perdere dati, registrati separandoli con il carattere virgola, se la lista non è congrua alla situazione del file.

GET#lfn,lista

legge dal file, aperto in lettura con numero logico lfn, un carattere per volta. È opportuno che la lista contenga solo variabili stringa.

CLOSElfn

chiude il file aperto con numero logico lfn. Se il file è di scrittura, viene scritto l'ultimo blocco registrando il carattere di EOF (End Of File), e viene aggiornata la Directory. Se il file è di lettura viene chiuso senza modificare la situazione del dischetto, ma risulta ugualmente necessario chiuderlo.

Riportiamo il programma PRFSDISCO1, come esempio. Questo programma è stato ottenuto modificando il programma PRFL1C, relativo alla cassetta, per mostrare come la scrittura di un file sequenziale avviene nello stesso modo anche sul dischetto.

```
10 REM PRFSDISCO1
15 REM CREAZIONE FILE SEQUENZIALE
20 REM RECORD LOGICO FORMATO DA 3 CAMPI
25 REM CAMPI DI LUNGHEZZA VARIABILE
30 REM A$←COGNOME
35 REM B$←NOME
40 REM C$←DATO NUMERICO LETTO COME STRINGA
45 REM SEPARATORE DI CAMPO = CHR$(13)
50 REM UN SOLO PRINT SCRIVE IL RECORD
55 REM SENZA PUNTEGGIATURA FINALE
60 REM IL SISTEMA AGGIUNGE CHR$(13)
65 REM ALLA FINE DEL RECORD
70 REM ";" COME SEPARATORE TRA LE VARIABILI
75 REM PER NON AGGIUNGERE SPAZI
```

```
80 CH$=CHR$(13):OPEN2,8,2,"FSDISCO1,S,W"
85 V$=CHR$(44):PV$=CHR$(59)
90 CD$="[illegible]"
95 PRINT"[illegible]":INPUT"COGNOME:  ";A$
100 INPUT"NOME:  ";B$
105 INPUT"NUMERO:";C$
110 PRINT#2,A$;CH$;B$;CH$;C$
115 PRINT#2,A$,CH$,B$,CH$,C$
120 PRINT#2,A$;V$;B$;V$;C$
125 PRINT#2,A$;PV$;B$;PV$;C$
130 R$="":PRINT CD$;"ALTRO RECORD (S/N)?";
135 GETR$:IFR$=""THEN135
140 IFR$<>"S"THEN155
145 GOTO95
155 CLOSE2
160 OPEN2,8,2,"FSDISCO1,S,R":OPEN4,4
165 PRINT#4,"FILE LETTO CON GET#":PRINT#4
170 GET#2,R$:TS=ST:PRINT#4,R$;"*";
175 IFTS<>64THEN170
180 PRINT#4:CLOSE2:CLOSE4
195 OPEN2,8,2,"FSDISCO1,S,R":OPEN4,4
200 PRINT#4,"FILE LETTO CON INPUT#":PRINT#4
205 INPUT#2,R$:TS=ST:PRINT#4,R$
210 IFTS<>64THEN205
215 PRINT#4:CLOSE2:CLOSE4:STOP
```

Abbiamo fatto girare il programma caricando due record identici a quelli usati per provare il corrispondente programma PRFL1C e abbiamo ottenuto gli stessi risultati, che seguono, senza ripetere i commenti, per i quali ti rimandiamo al Paragrafo 2.2. Ti raccomandiamo di confrontare tra loro i due programmi, per rilevare le differenze nella stesura delle operazioni relative al trattamento dei file.

```
FILE LETTO CON INPUT#

PRIMOC
PRIMON
11111
PRIMOC
PRIMON
11111
PRIMOC
```

```
PRIMOC;PRIMON;11111
SECONDOC
SECONDON
22222
SECONDOC
SECONDON
22222
SECONDOC
SECONDOC;SECONDON;22222
```

FILE LETTO CON GET#

```
P*R*I*M*O*C*
*P*R*I*M*O*N*
*1*1*1*1*1*
*P*R*I*M*O*C* * * * * * * * * * *
* * * * * * * * * * *P*R*I*M*O*N* * * * * * * * * * *
* * * * * * * * * * *1*1*1*1*1*
*P*R*I*M*O*C*,*P*R*I*M*O*N*,*1*1*1*1*1*
*P*R*I*M*O*C*;*P*R*I*M*O*N*;*1*1*1*1*1*
*S*E*C*O*N*D*O*C*
*S*E*C*O*N*D*O*N*
*2*2*2*2*2*
*S*E*C*O*N*D*O*C* * * * * * * * * * *
* * * * * * * * * * *S*E*C*O*N*D*O*N* * * * * * * * * * *
* * * * * * * * * * *2*2*2*2*2*
*S*E*C*O*N*D*O*C*,*S*E*C*O*N*D*O*N*,*2*2*2*2*2*
*S*E*C*O*N*D*O*C*;*S*E*C*O*N*D*O*N*;*2*2*2*2*2*
*
```

Riportiamo le due istruzioni 205 e 206, da usare per modificare il programma PRFSDISCO1, per non perdere campi nel record che usa, come separatore registrato, la virgola.

```
205 INPUT#2,R1$,R2$,R3$:TS=ST
206 PRINT#4,R1$:PRINT#4,R2$:PRINT#4,R3$
```

Per proporre un esempio di file sequenziale numerico, riportiamo il programma PRFSDISCO2, nel quale scriviamo come nell'esempio PRFL2C della cassetta, quattro record numerici formati da due campi ciascuno. Dopo aver chiuso il file lo apriamo in lettura, leggiamo con INPUT# e stampiamo i dati e poi, in una seconda fase, leggiamo con GET# e stampiamo i dati carattere per carattere. In questo caso non andiamo a leggere il buffer, che si trova nella RAM dell'unità 1541.

```
10 REM PRFSDISCO2
15 REM CARICAMENTO FILE NUMERICO
20 REM OGNI RECORD LOGICO COMPRENDE 2 CAMPI
25 REM IL PRIMO E' UN NUMERO INTERO
30 REM IL SECONDO UN NUMERO DECIMALE
35 REM SI USA CHR$(13) COME SEPARATORE DI CAMPO
40 REM PER USCIRE DARE UN INTERO NULLO
45 REM IL RETURN FINALE LO METTE IL SISTEMA
50 CH$=CHR$(13)
55 OPEN2,8,2,"FSDISCO2,S,W":OPEN4,4
60 PRINT"NUMERI INTERI <= 32767"
65 PRINT"PER USCIRE RISPONDERE 0 (ZERO) A INTERO"
70 INPUT"INTERO: ";I%
75 IFI%=0THENCLOSE2:GOTO110
80 INPUT"DECIMALE: ";D
85 PRINT#2,I%;CH$;D
90 GOTO70
110 REM LETTURA FILE NUMERICO
115 REM CON INPUT VENGONO LETTI I 2 CAMPI
120 REM ESSI VENGONO STAMPATI
125 REM QUANDO E' FINITO IL FILE VIENE CHIUSO
130 REM E POI VIENE RIAPERTO, LETTO CON GET#
133 REM E STAMPATO CARATTERE PER CARATTERE
135 OPEN2,8,2,"FSDISCO2,S,R"
145 PRINT#4:PRINT#4,"CONTENUTO FILE NUMERICO"
150 PRINT#4
155 INPUT#2,I%,D:TS=ST
160 PRINT#4,I%,D
165 IFTS<>64THEN155
175 CLOSE2:PRINT#4
180 OPEN2,8,2,"FSDISCO2,S,R":I=0
183 PRINT#4,"FILE LETTO CON GET#":PRINT#4
185 GET#2,A$:TS=ST
190 PRINT#4,ASC(A$+CHR$(0));
193 I=I+1:IFI=8THENPRINT#4:I=0
195 IFTS=64 THEN CLOSE2:PRINT#4:CLOSE4:STOP
200 GOTO185
```

Riportiamo i risultati del programma.

```
CONTENUTO FILE NUMERICO

 123              -678.43
-67               543.78
 3456               90.67
-43               890.5

FILE LETTO CON GET#

 32  49  50  51  32  13  45  54
 55  56  46  52  51  32  13  45
 54  55  32  13  32  53  52  51
 46  55  56  32  13  32  51  52
 53  54  32  13  32  57  48  46
 54  55  32  13  45  52  51  32
 13  32  56  57  48  46  53  32
 13
```

Come puoi vedere, paragonando questi risultati con quelli del file su cassetta (abbiamo scritto campi identici) la registrazione avviene nello stesso modo.

Nei due esempi di programma puoi trovare applicati i comandi che abbiamo trattato in questo paragrafo. Non abbiamo fatto l'analisi degli errori tramite il canale 15, dato che si tratta solo di esempi. Ti raccomandiamo, però, di non tralasciarla in un programma elaborativo.
Osserva alla linea 190 la sequenza:
PRINT#4,ASC(A$+CHR$(0))
è necessario sommare alla stringa A$ la stringa CHR$(0) per ovviare all'inconveniente che, se A$ è la stringa nulla, la funzione ASC dà errore.

Chiudiamo questo paragrafo con il programma esempio APRISEQUENZIALI, che dimostra quanti file sequenziali possono essere aperti contemporaneamente sul dischetto.

```
1 REM APRISEQUENZIALI
5 OPEN15,8,15,"I"
7 INPUT"QUANTI FILE SEQ.";N
15 ONNGOTO60,50,40,30,20
20 OPEN6,8,6,"@0:SEQ5,S,W":GOSUB100
21 PRINT#6,"RECORDSEQ5":GOSUB100
30 OPEN5,8,5,"@0:SEQ4,S,W":GOSUB100
31 PRINT#5,"RECORDSEQ4":GOSUB100
40 OPEN4,8,4,"@0:SEQ3,S,W":GOSUB100
41 PRINT#4,"RECORDSEQ3":GOSUB100
50 OPEN3,8,3,"@0:SEQ2,S,W":GOSUB100
51 PRINT#3,"RECORDSEQ2":GOSUB100
60 OPEN2,8,2,"@0:SEQ1,S,W":GOSUB100
61 PRINT#2,"RECORDSEQ1":GOSUB100
65 ONNGOTO74,73,72,71,70
70 CLOSE6:GOSUB100
71 CLOSE5:GOSUB100
72 CLOSE4:GOSUB100
73 CLOSE3:GOSUB100
74 CLOSE2:GOSUB100
99 CLOSE15:STOP
100 INPUT#15,EN,EM$,ET,ES
110 PRINTEN;EM$;ET;ES
120 RETURN
```

Prova a far girare il programma; vedrai che dopo aver aperto 3 file ricevi la segnalazione di errore : 70 NO CHANNEL.
Il programma chiede quanti file vuoi aprire, li apre, uno dopo l'altro e poi li chiude; sul video compare il messaggio di errore o segnalazione richiesto con il sottoprogramma in 100.

Ti segnaliamo un errore del DOS, che è bene tener presente. Quando si scrive un file sequenziale, formato da record logici di 254 caratteri, compresi i fine campo, cioè con record logico coincidente con il settore, succede che dopo la chiusura del file, si perde un settore.

Abbiamo provato il programma SEQ254, che segue:

```
10 REM SEQ254
15 REM R1$ CONTIENE 70 ASTERISCHI
16 REM R2$ CONTIENE 55 ASTERISCHI
20 R1$="":FORK=1TO70:R1$=R1$+"*":NEXTK
21 R2$="":FORK=1TO55:R2$=R2$+"*":NEXTK
25 OPEN15,8,15,"I"
30 OPEN2,8,2,"PSEQ254,S,W"
35 FORK=1TO4:PRINT#2,R1$;CHR$(13);R2$;CHR$(13);
36 PRINT#2,R1$;CHR$(13);R2$;CHR$(13);
40 NEXTK
45 CLOSE2:CLOSE15
```

esso scrive 4 record formati ciascuno da 4 campi, per un totale di 254 caratteri. Dopo la chiusura del file abbiamo listato la directory del dischetto, che prima della prova è stato formattato, che riportiamo.

```
0 [illegible]
4     "PSEQ254"          SEQ
659 BLOCKS FREE.
```

Come vedi manca un blocco nel computo. Abbiamo eseguito sul dischetto la funzione VALIDATE, e abbiamo nuovamente listato la directory. Come vedi il blocco è stato ricuperato.

```
0 [illegible]
4     "PSEQ254"          SEQ
660 BLOCKS FREE.
```

Poi abbiamo provato a leggere il file sequenziale con il programma LEGGI-SEQ254, che segue.

```
10 REM LEGGISEQ254
15 REM LEGGE E STAMPA CONTENUTO FILE PSEQ254
20 K=1
25 OPEN15,8,15,"I"
30 OPEN2,8,2,"PSEQ254,S,R"
35 INPUT#2,R$:TS=ST
36 PRINT K;R$:K=K+1
40 IFTS=64THENCLOSE2:CLOSE15:STOP
45 GOTO35
```

Il file viene letto tutto senza perdita di record logici.

2.2 ESEMPIO DI ARCHIVIO SEQUENZIALE

Abbiamo realizzato su disco un programma analogo a PRFL3C, mantenendone inalterate molte caratteristiche, onde consentire un confronto tra i due tipi di supporti.

Anche in questo caso abbiamo un record logico di lunghezza variabile, formato da 6 campi di lunghezza variabile. I campi sono: COGNOME, NOME, TELEFONO, INDIRIZZO, CAP e CITTA'. Non abbiamo controllato la lunghezza dei campi (tagliandoli se superano singolarmente i 79 caratteri), ma, stando alla loro natura, sembra difficile superare 79 caratteri, a meno di non farlo volutamente. In quest'ultimo caso si avrebbe segnalazione di errore in lettura.

Non si ha controllo sul numero di record caricabili; ovviamente se si scrivono moltissimi record si può superare la capacità del dischetto.
Il programma non presenta un menù iniziale, ma pone successivamente domande sull'operazione da svolgere. Le operazioni possibili sono:

1) Crea il file ex-novo
2) Lista il file sulla stampante
3) Aggiorna il file consentendo di: .inserire record,
.modificare record,
.cancellare record.

Ogni fase conclude il programma e devi ripartire con RUN per eseguire un'altra operazione. Anche le tre sottofasi della fase 3, possono essere svolte una alla volta.

Non riportiamo qui lo schema a grandi blocchi del programma, in quanto resta valido quello della Figura 3.1 del quarto Volume.

Abbiamo mantenuto la tecnica dei sottoprogrammi; in conseguenza il programma è facilmente modificabile.
Per uscire dalla richiesta dei dati, devi rispondere con "*" alla richiesta del cognome.

La più evidente differenza, rispetto all'analogo programma per la cassetta, sta nella fase dell'aggiornamento. Infatti per aggiornare il file si procede così:

- si apre un file di nome TEMP, sempre di tipo sequenziale, per scrittura, sul quale vengono via via ricopiati i record letti dal file da aggiornare,
- per inserire, devono essere forniti i nuovi record, sempre in ordine di cognome e nome, il file in lettura e quello in scrittura vengono fatti avanzare parallelamente, in modo che le inserzioni vadano al posto giusto, scartando dati forniti fuori ordine,
- per modificare record la procedura è analoga alla precedente,
- per cancellare record, il record non viene ricopiato,
- alla fine, dopo aver chiuso i due file, viene cancellato il file sorgente e viene cambiato nome al file TEMP, assegnandogli il nome del file sorgente.

Riportiamo nella Figura 2.1 uno schema a grandi blocchi della fase di aggiornamento per inserzione.

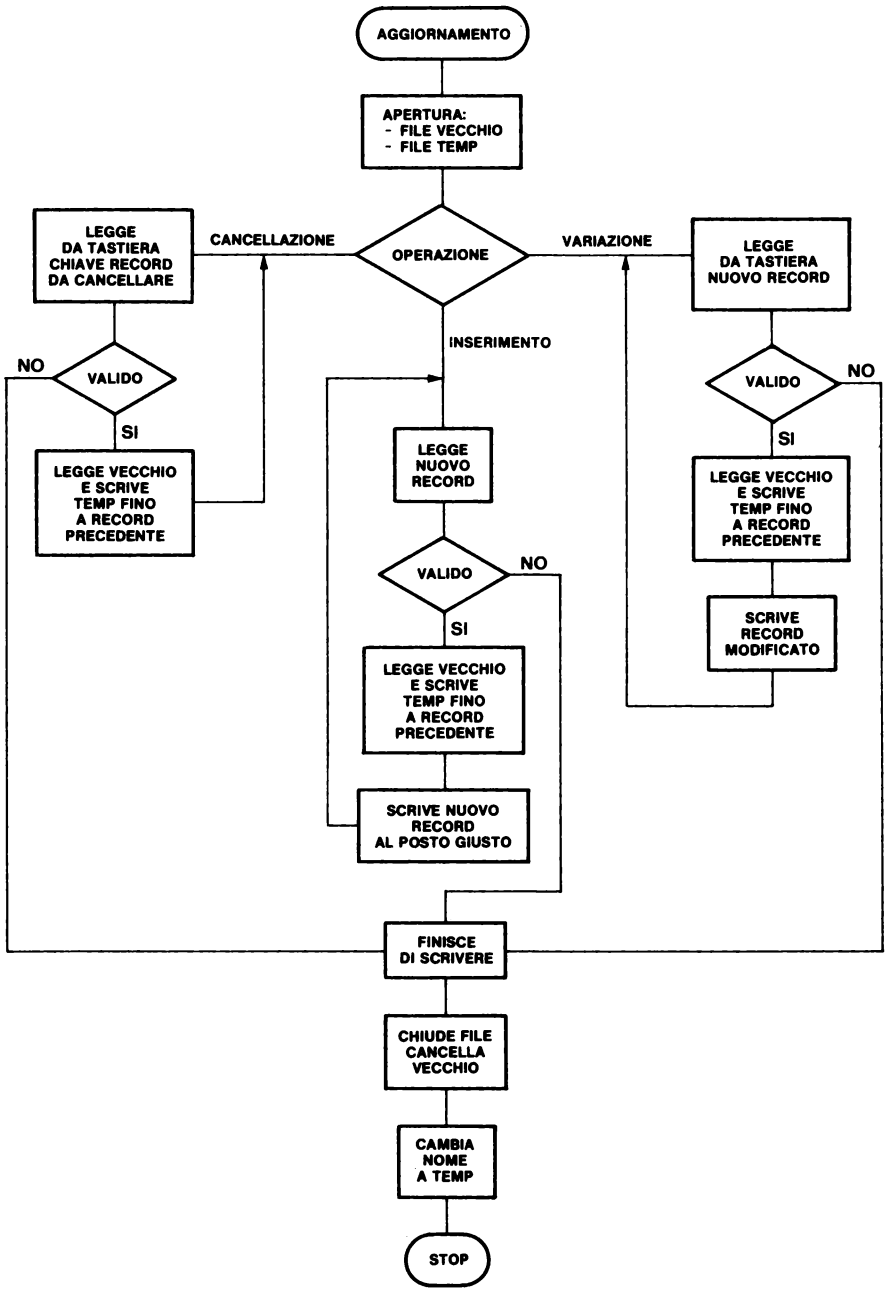


Fig. 2.1 Aggiornamento per inserzione

Ti consigliamo di dedicare un pò di tempo all'analisi di questa parte del programma; l'aggiornamento di un file sequenziale non è mai un'operazione semplice, se si vuole mantenere un ordinamento nei record.

Passiamo ad elencare le variabili e le costanti usate nel programma.

PR=4, numero periferica di stampa
NF=4, lfn del file di stampa
CH$=CHR$(13), carattere separatore RETURN
SP$=" ", 4 spazi
I$(6), vettore per leggere i dati da tastiera
I1$(6), vettore per leggere il record da disco in aggiornamento
D$(6), vettore per le descrizioni dei campi
R$, variabile per le risposte
LC$, vecchio cognome
LN$, vecchio nome
FS, memorizzazione parola di stato ST
EN, EM$, ET, ES, variabili controllo errore
SW, controllo dati ingresso, 0 per si, 1 per finiti i dati
FF, switch per fine file, 1 finito file input
FL, switch utilizzo record letto per aggiornamento, 0 utilizzato, 1 da utilizzare
R, K variabili di controllo

Segue il listato del programma.

```
10 REM PRFSDISC03
12 REM CREAZIONE E GESTIONE FILE SEQUENZIALE
14 REM PARAMETRI DI STAMPA-APERTURA CANALE 15
16 NF=4:PR=4:OPEN15,8,15:GOSUB362
18 REM COSTANTI E VARIABILI
20 CH$=CHR$(13):SP$="    "
22 DIMI$(6),I1$(6),D$(6)
24 D$(1)="COGNOME ":D$(2)="NOME ":D$(3)="TEL. "
26 D$(4)="INDIRIZZO ":D$(5)="CAP ":D$(6)="CITTA' "
28 REM SCELTA OPERAZIONE
30 PRINT"]CREAZIONE FILE  S/N ";:INPUTR$
32 IFR$<>"S"THEN70
34 REM----------------
36 REM CREAZIONE FILE
38 REM----------------
40 GOSUB278:GOSUB266:GOSUB282
42 PRINT#15,"I0":REM INIZIALIZZAZIONE
```

```
44 REM APERTURA FILE PER SCRIVERE-CANALE 2
46 OPEN2,8,2,"0:"+NF$+",S,W":GOSUB362
48 LC$="":LN$="":REM VECCHIO COGNOME E NOME
50 GOSUB228:REM RICHIESTA DATI
52 IFSW=0THEN60
54 CLOSE2:CLOSE15
56 PRINT"■FINITO CARICAMENTO    FILE":STOP
58 REM CONTROLLO ORDINE RECORD
60 IFI$(1)>LC$THEN66
62 IFI$(1)=LC$THENIFI$(2)>LN$THEN66
64 GOSUB272:GOTO50
66 LC$=I$(1):LN$=I$(2)
68 GOSUB256:GOTO50
70 REM-------------
72 REM STAMPA FILE
74 REM-------------
76 PRINT"□STAMPA FILE S/N";:INPUTR$
78 IFR$<>"S"THEN104
80 GOSUB278:GOSUB266
82 PRINT#15,"I0":REM INIZIALIZZAZIONE
84 GOSUB282:GOSUB286
86 OPENNF,PR:PRINT#NF,"LISTA ARCHIVIO ";NF$
88 PRINT#NF
90 REM LETTURA E STAMPA RECORD
92 GOSUB250:PRINT#NF,I$(1);SP$;I$(2)
94 PRINT#NF,I$(3):PRINT#NF,I$(4)
96 PRINT#NF,I$(5);SP$;I$(6):PRINT#NF
98 IFFS<>64THEN 90
100 CLOSE3:CLOSENF:CLOSE15
102 PRINT"■FINITO LISTA":STOP
104 REM--------------------
106 REM AGGIORNAMENTO FILE
108 REM--------------------
110 FF=0:PRINT"■AGGIORNAMENTO FILE"
112 GOSUB278:GOSUB266
114 PRINT#15,"I0":REM INIZIALIZZAZIONE
116 GOSUB282:GOSUB286
118 REM APERTURA FILE TEMPORANEO PER SCRIVERE
120 OPEN2,8,2,"0:TEMP,S,W":GOSUB362
122 LC$="":LN$=""
124 PRINT"□TIPO DI VARIAZIONE:"
126 PRINT"1=INSERIMENTO":PRINT"2=VARIAZIONE"
```

```
128 PRINT"3=CANCELLAZIONE"
130 INPUTR:IFR<1ORR>3THEN130
132 FL=0:REM 1 SE ULT. REC. LETTO DA SCRIVERE
134 FF=0:REM 1 SE FINITO FILE INPUT
136 ONRGOTO138,180,202:REM SCELTA OPERAZIONE
138 REM--------------------
140 REM INSERIMENTO RECORD
142 REM--------------------
144 GOSUB228:IFSW=0THEN156
146 GOSUB328:CLOSE3:CLOSE2
148 PRINT"FINITO AGGIORNAMENTO"
150 REM SISTEMAZIONE FILE SU DISCO
152 PRINT#15,"S0:"+NF$
154 PRINT#15,"R0:"+NF$+"=TEMP":CLOSE15:STOP
156 IFI$(1)>LC$THEN170:REM INSERISCE
158 REM NOMI IN DISORDINE
160 IFI$(1)<LC$THENGOSUB272:GOTO138
162 REM COGNOMI UGUALI
164 IFI$(2)>LN$THEN170
166 IFI$(2)<LN$THENGOSUB272:GOTO138
168 GOSUB262:GOTO138
170 REM NOMI IN ORDINE
172 LC$=I$(1):LN$=I$(2):GOSUB292
174 IFI$(1)<>I1$(1)THEN178
176 IFI$(2)=I1$(2)THENGOSUB262:GOTO138
178 GOSUB256:GOTO138
180 REM------------------
182 REM MODIFICA RECORD
184 REM------------------
186 GOSUB228:IFSW=0THEN190
188 GOTO146
190 IFI$(1)>LC$THEN196
192 IFI$(1)=LC$THENIFI$(2)>LN$THEN196
194 GOSUB272:GOTO180
196 LC$=I$(1):LN$=I$(2):GOSUB340
198 IFFL=1ANDFF=0THEN180
199 IFFL=1ANDFF=1THENFL=0:GOTO146
200 GOSUB256:IFFF=1THEN146
201 GOTO180
202 REM----------------------
204 REM CANCELLAZIONE RECORD
206 REM----------------------
```

```
208 PRINTI$(1);:INPUTI$(1):IFI$(1)="*"THEN146
210 PRINTD$(2);:INPUTI$(2):IFI$(1)>LC$THEN216
212 IFI$(1)=LC$THENIFI$(2)>LN$THEN216
214 GOSUB272:GOTO202
216 LC$=I$(1):LN$=I$(2):GOSUB340
218 IFFF=1ANDFL=0THEN146
220 GOTO202
222 REM-----------------
224 REM SOTTOPROGRAMMI
226 REM-----------------
228 REM RICHIESTA DATI
230 SW=0:REM 1 SE FINITI DATI
232 PRINT"▐ 1 ";D$(1);:INPUTI$(1)
234 IFI$(1)="*"THENSW=1:RETURN
235 FORK=2TO6:I$(K)=CHR$(160):NEXTK
236 FORK=2TO6:PRINTK;D$(K);:INPUTI$(K):NEXTK
238 PRINT"▐CONFERMI S/N";:INPUTR$
240 IFR$="S"THENRETURN
242 PRINT"QUALE CAMPO ";:INPUT R
244 IFR<1ORR>6THENPRINT"▐";:GOTO242
246 PRINTD$(R);:I$(R)=CHR$(160):INPUTI$(R)
247 PRINT"▐";
248 FORK=1TO6:PRINTK;D$(K);I$(K):NEXTK:GOTO238
250 REM LETTURA RECORD DA DISCO
252 FORK=1TO6:INPUT#3,I$(K):FS=ST
254 GOSUB362:NEXTK:RETURN
256 REM SCRITTURA NUOVO RECORD
258 FORK=1TO6:PRINT#2,I$(K);CH$;
260 GOSUB362:NEXTK:RETURN
262 REM MESSAGGIO NOMI UGUALI
264 PRINT"NOMI UGUALI":GOSUB266:RETURN
266 REM ATTESA TASTO
268 GETA$:IFA$=""THEN266
270 RETURN
272 REM MESSAGGIO FUORI ORDINE
274 PRINT"▐NOMI FUORI ORDINE":GOSUB266
276 RETURN
278 REM RICHIESTA DISCO DATI
280 PRINT"▐MONTA DISCO DATI":RETURN
282 REM RICHIESTA NOME FILE
284 INPUT"NOME FILE ";NF$:RETURN
286 REM APERTURA FILE PER LEGGERE-CANALE 3
```

```
288 OPEN3,8,3,"0:"+NF$+",S,R":GOSUB362
290 RETURN
292 REM LEGGE RECORD E SCRIVE FINO AL
294 REM RECORD PRECEDENTE SU TEMP
296 IFFF=1ANDFL=0THENRETURN
298 IFFL<>0THEN304
300 GOSUB316
302 IFFS=64THENFF=1
304 IFI1$(1)<I$(1)THEN310
306 IFI1$(1)=I$(1)THENIFI1$(2)<I$(2)THEN310
308 FL=1:RETURN
310 GOSUB322:FL=0
312 IFFF=1THENRETURN
314 GOTO300
316 REM LETTURA RECORD
318 FORK=1TO6:INPUT#3,I1$(K):FS=ST
320 GOSUB362:NEXTK:RETURN
322 REM SCRITTURA RECORD
324 FORK=1TO6:PRINT#2,I1$(K)
326 GOSUB362:NEXTK:RETURN
328 REM SCRIVE FILE TEMP FINO ALLA FINE
330 REM DEL FILE DI INPUT
332 IFFF=1ANDFL=0THENRETURN
334 IFFL<>0THENFL=0:GOSUB322:GOTO328
336 GOSUB316:GOSUB322:IFFS=64THENFF=1:RETURN
338 GOTO336
340 REM COPIA FILE FINO AL RECORD CERCATO
342 IFFF=1ANDFL=0THEN355
344 IFFL<>0THEN348
346 GOSUB316:IFFS=64THENFF=1
348 IFI1$(1)<I$(1)THEN356
350 IFI1$(1)=I$(1)THENIFI1$(2)<I$(2)THEN356
352 IFI1$(1)=I$(1)ANDI1$(2)=I$(2)THENFL=0:RETURN
354 FL=1
355 PRINT"NON TROVATO":GOSUB266:RETURN
356 GOSUB322
358 IFFF=1THEN355
360 GOTO346
362 REM ROUTINE ERRORE
364 INPUT#15,EN,EM$,ET,ES
366 IFEN=0THEN RETURN
368 PRINT"ERRORE DISCO"
370 PRINTEN,EM$,ET,ES:STOP
```

Segue un pezzo del listato del file prova, che noi abbiamo chiamato FSDISCO3.

```
LISTA ARCHIVIO FSDISCO3

PRIMO     SIGNORE
1111111
VIA PRIMA 1
12345     MILANO

SECONDO     SIGNORE
222222
VIA SECONDA 2
65432     MILANO

TERZO     SIGNORE
333333
VIA TERZA 3
98765     MILANO
```

Nel programma non è stata particolarmente curata la lista dei record del file. I dati sono stampati su 4 linee, ponendo sulla prima il cognome e il nome, sulla seconda il telefono, sulla terza l'indirizzo e sulla quarta il CAP e la città. Non è stato fatto il controllo della lunghezza del foglio. Potrai aggiungere tu tutte queste cose, come utile esercizio.

Nel listato abbiamo messo in rilievo con REM opportune le diverse parti del programma, e, per questa ragione, non passiamo ad elencare le funzioni dei diversi gruppi di linee.

Preferiamo terminare con dei suggerimenti per migliorare il programma. La fase dell'aggiornamento potrebbbe essere impostata in modo che:

- viene chiesto un cognome e nome,
- viene chiesto se si vuole: correggere, inserire o cancellare,
- si fanno avanzare i file fino al punto giusto,
- si esegue l'operazione richiesta,
- si procede fino ad esaurire le richieste, che, ovviamente, devono essere passate in ordine di cognome e nome,
- si finisce di ricopiare il file, se necessario, e si chiude.

Con questa impostazione, le tre operazioni di aggiornamento potrebbbero essere svolte in una sola fase.

CAPITOLO 3

FILE RANDOM

3.1 FILE RANDOM DI DATI

In questo contesto il file RANDOM è un file nel quale si accede ai singoli record conoscendo l'indirizzo fisico del settore al quale appartengono.
Manteniamo la solita distinzione tra record logico e record fisico. Il record logico è quello che interessa la logica del programma, ed è formato dai caratteri che servono per contenere i sui campi. Il record fisico è il settore del dischetto. Il programmatore deve aver cura di strutturare il suo record logico in modo di ADATTARLO alla struttura fisica sulla quale si deve appoggiare, per renderne il più semplice possibile la gestione.
Sarà quindi buona norma, dopo aver studiato il tracciato del record logico, aggiungere eventualmente qualche campo o qualche carattere a qualche campo già definito, in modo che la lunghezza del record logico possa:

- coincidere con il settore,
- essere un sottomultiplo del settore,
- essere un multiplo della lunghezza di un settore.

Noi, negli esempi, usiamo questa tecnica, e creiamo, insieme al file principale random, anche un file indice di tipo sequenziale.

Le elaborazioni sono più semplici se si lavora con record di lunghezza fissa, ma, dopo essersi impadroniti a fondo dell'argomento, è possibile anche lavorare con record di lunghezza variabile, ponendo come primo campo di ogni record l'informazione circa la lunghezza del record stesso.

Prendiamo ora in esame i comandi del DOS che consentono la gestione dei file RANDOM.

Per operare sui file random vengono impegnati due canali, il canale 15 dei comandi, e un altro canale (da 2 a 14) con relativo buffer. Le operazioni si svolgono in due tempi:

- SCRITTURA: le normali operazioni **PRINT#** scrivono nel buffer; una particolare operazione, lanciata sul canale comandi, trasferisce il buffer in un ben determinato settore del dischetto.
- LETTURA: una particolare operazione, lanciata sul canale comandi, trasferisce un ben determinato settore del dischetto nel buffer; le normali operazioni **INPUT#** e **GET#** leggono i dati dal buffer.

Per far partire una operazione su file sono necessarie due OPEN, una sul canale 15, e una sull'altro canale, con una stringa comando speciale.

OPEN15,8,15

apre il canale comandi, usando, secondo la nostra abitudine lfn=15.

OPENlfn,8,sa,"#"

apre il canale sa (che può avere un valore da 2 a 14), con il numero logico lfn (che può avere il valore da 1 a 127). Noi, come al solito, per semplicità, usiamo lo stesso numero per lfn e sa, da 2 a 14, escludendo, di solito, il 4, che usiamo per la stampante.
La stringa speciale "#" può essere scritta facendo seguire al carattere # un numero, il numero del buffer che si vuole utilizzare, per esempio "#3". Se richiedi un particolare buffer e questo è occupato ottieni il messaggio di errore: NO CHANNELS. Lo stesso messaggio compare se si cercano di aprire più file di quelli consentiti.
Se vuoi sapere quale buffer ti ha assegnato il sistema (avendo usato solo "#"), devi eseguire, subito dopo la OPEN, un'istruzione GET#lfn.
L'istruzione OPEN non deve precisare se si vuole leggere o scrivere; infatti in questo caso si possono fare ambedue le operazioni. La OPEN mette a disposizione un canale e un buffer; a questi si fa, come al solito, riferimento, citando il numero lfn.

Vediamo ora i comandi che si possono passare al DOS, come stringa-comando, con una operazione PRINT# sul canale 15; essi sono riportati nella Tabella 3.1.

Tab. 3.1 COMANDI PER IL DOS

COMANDI		
Completi	Abbreviati	Formato Stringa
BLOCK-READ	B-R	"B-R:"ch,dr,t,s
BLOCK-WRITE	B-W	"B-W:"ch,dr,t,s
BLOCK-EXECUTE	B-E	"B-E:"ch,dr,t,s
BUFFER-POINTER	B-P	"B-P:"ch,p
BLOCK-ALLOCATE	B-A	"B-A:"dr,t,s
BLOCK-FREE	B-F	"B-F:"dr,t,s
Memory-Write	M-W	"M-W"adl/adh/nc/data
Memory-Read	M-R	"M-R"adl/adh
Memory-Execute	M-E	"M-E"adl/adh
USER	U	"Ui:parms"

dove:

ch, numero canale, sa della OPEN... "#"
dr, numero unità: 0
t, numero traccia, da 1 a 35
s, numero settore, da 0 a 20, o meno, a seconda della traccia
p, posizione del puntatore nel buffer
adl, byte basso del blocco di memoria
adh, byte alto del blocco di memoria
nc, numero caratteri da trasferire, da 1 a 34
data, dato attuale in notazione esadecimale, si deve usare la funzione CHR$ del codice decimale equivalente
i, numero di riferimento nella tabella USER
parms, parametri associati a U

Nella stringa dei comandi devono essere usate le abbreviazioni per i comandi Memory, per gli altri si possono anche usare i nomi completi.
I parametri, se sono costanti, possono essere scritti all'interno delle virgolette, se sono variabili, no.
All'interno delle virgolette possono essere usati come caratteri separatori lo spazio e la virgola; all'esterno delle virgolette, solo il punto e virgola.

Nella Tabella 3.2 sono specificati i comandi USER.

Tab. 3.2 SALTI USER ALLA MEMORIA DELL'UNITA' 1541

TABELLA DEI SALTI		
Prima Definizione	Definizione Alternativa	Significato Comando
U1	UA	sostituisce BLOCK-READ
U2	UB	sostituisce BLOCK-WRITE
U3	UC	salto a 1280 (0500H)
U4	UD	salto a 1283 (0503H)
U5	UE	salto a 1286 (0506H)
U6	UF	salto a 1289 (0509H)
U7	UG	salto a 1292 (050CH)
U8	UH	salto a 1295 (050FH)
U9	UI	salto a 65530 (FFFAH)
U0	UJ	salto alla routine di accensione

Esaminiamo ora separatamente ogni comando, trattando prima il gruppo che si riferisce alla gestione dei file di dati, e dopo quello che interviene sulla programmazione dell'unità 1541. Non riportiamo il formato di ogni comando, rilevabile dalla Tabella 3.1.

BLOCK-READ, abbreviazione facoltativa B-R

Trasferisce il settore specificato nel buffer assegnato al momento della OPEN. Mantiene l'indicazione sulla posizione del puntatore, registrata nell'operazione di scrittura del blocco, cioè della posizione del puntatore dopo l'ultima PRINT# effettuata prima di trasferire il blocco su disco. Tale indicazione sta nella posizione

0 del buffer. Dopo il trasferimento nel buffer effettuato con B-R, se, dopo INPUT# o GET#, si analizza la parola di stato ST, quando viene raggiunta la posizione finale del puntatore si trova ST=64.

BLOCK-WRITE, abbreviazione facoltativa B-W

Trasferisce il contenuto del buffer nel settore specificato, registrando la posizione raggiunta dal puntatore con l'ultima PRINT# nella posizione 0, ma predisponendo il puntatore sulla posizione 1. Se, dopo, la lettura del blocco viene effettuata con B-R e non si sposta il puntatore, il primo carattere disponibile è quello di posizione 1, mentre se viene effettuata con U1 e non si sposta il puntatore, il primo carattere disponibile è quello di posizione zero.

BUFFER-POINTER, abbreviazione facoltativa B-P

Consente di spostare il puntatore all'interno del buffer, sia dopo avere ricevuto in esso dati in seguito a B-R o a U1, che quando si sta riempiendolo di dati prima di eseguire B-W o U2. La posizione del puntatore è molto importante, perchè le operazioni GET#, INPUT# e PRINT# agiscono a partire dalla sua posizione. Il puntatore può avere valori da 0 (primo carattere del buffer) a 255 (ultimo carattere del buffer). Se, quando usi i file random, vuoi rispettare la struttura degli altri file, non devi usare per i dati i primi due caratteri del buffer, usati negli altri casi dal sistema per concatenare tra loro i settori del disco. Quindi, prima di iniziare a scrivere, devi con B-P posizionare a 2 il puntatore, puoi, in conseguenza, usare per i dati solo 254 caratteri. Se hai operato in scrittura in questo modo, devi analogamente posizionare a 2 il puntatore prima di cominciare a leggere i dati.
Supponendo di usare un file random con il record logico di 127 caratteri, possiamo memorizzare in un settore due record logici; il primo inizia con il puntatore al valore 2, il secondo con il puntatore al valore 129.
Operando sul puntatore si può accedere con la massima libertà ai campi di un record, si deve, però fare attenzione a quale comando si usa per memorizzare il buffer su disco. Infatti, come abbiamo già visto, B-W mantiene traccia dell'ultima posizione del puntatore; vedrai che U2 non lo fa.

BLOCK-ALLOCATE, abbreviazione facoltativa B-A

registra nella BAM l'occupazione del settore specificato. Prima di poter disporre di un settore è necessario sapere se è disponibile; questo si ottiene con B-A, chiedendo di allocare un settore e analizzando dopo la situazione di errore, tramite il canale 15. Se la variabile EN contiene 65, corrispondente al messaggio NO BLOCK, significa che il settore è già occupato; ma, fortunatamente, in questo caso ET e ES contengono gli indirizzi di traccia e settore del prossimo settore disponibile. Basta usare nuovamente B-A con gli indirizzi forniti da ET e ES per allocare un settore

disponibile e renderlo indisponibile. Se EN=65 e ET=0, significa che il dischetto è completamente pieno.
Se, invece, EN contiene 0, significa che il settore richiesto è stato allocato. Riportiamo, più avanti, la routine ALLOCA che esegue correttamente questa operazione, senza andare ad usare settori, eventualmente liberi, della traccia 18.

BLOCLK-FREE, abbreviazione facoltativa B-F

libera un settore già occupato, modificando la BAM

È importante notare quanto segue:

B-A e B-F lavorano anche se il canale per l'accesso diretto al file non è aperto (cioe non è stata fatta la OPEN..."#"), ma questo è pericoloso; infatti la BAM viene riscritta quando si chiude un canale ad accesso diretto. Operando senza il canale dati aperto si rischia di non riscrivere la BAM aggiornata sul dischetto.

U1 (oppure UA)

Lavora come B-R, ma con il vantaggio che legge un intero blocco, senza tenere conto della posizione finale del puntatore nel buffer, prima dell'operazione di scrittura. Dopo U1, se non si muove il puntatore, il primo carattere disponibile è quello di posizione 0. Noi, in generale usiamo U1 per leggere, ma, in casi particolari può servire anche B-R.

U2 (oppure UB)

Lavora come B-W, ma non registra la posizione finale del puntatore nel buffer. Noi, di solito, scriviamo con U2.

Prima di passare alla descrizione dell'altro gruppo di comandi, riportiamo alcuni esempi. Se vuoi provare questi programmi devi operare così:

- prendi un dischetto nuovo, esegui l'operazione di formattazione e lo poni da parte per le prove;
- scrivi il programma esempio e lo memorizzi su un tuo dischetto per programmi, oppure carica dal dischetto dei programmi del libro il programma esempio;
- prima di dare RUN, spegni l'unità 1541, attendi un momento, poi riaccendila e monta il dischetto per le prove. Per vedere bene cosa succcede è necessario partire con la RAM dell'unità 1541 azzerata, altrimenti i buffer sono sporchi e questo può recare confusione.

Alcuni di questi programmi esempio lavorano su settori ad indirizzo fisso e si rischia di rovinare dischetti normali di lavoro.

Segue il programma RANDOMPROVE1; esso fa le seguenti cose:

- apre il canale 15, linea 40
- apre il canale 2 per i dati, linea 50
- scrive, senza intervenire sul puntatore, le tre stringhe A$, B$, C$ nel buffer, linea 60
- scrive con B-W il buffer nella traccia 20, settore 0, linea 70
- apre il canale 3 per i dati, linea 90
- scrive, senza intervenire sul puntatore, le tre stringhe A$, B$, C$ nel buffer, linea 100
- scrive con U2 il buffer nella traccia 20, settore 1
- chiude il canale 2 e il canale 3, linea 120.

Abbiamo usato due canali, aperti contemporaneamente, per ottenere di usare due buffer diversi.
Vengono scritti due settori, con gli stessi dati, usando i due comandi B-W e U2. Poi:

- trasferisce il settore 20,0 con il comando B-R nel buffer e, senza modificare il puntatore, legge i dati carattere per carattere con GET# e li stampa, linee da 125 a 150. Dai risultati vedi che la lettura con GET parte dal primo carattere del primo campo; 80 è il codice ASCII di P. Viene sentito il segnale EOF sul byte 20, l'ultimo stampato.

- trasferisce il settore 20,0 con il comando U1 nel buffer e, senza modificare il puntatore, legge i dati carattere per carattere con GET# e li stampa, linee da 155 a 180. Dai risultati vedi che la lettura con GET# parte questa volta dal carattere di posizione 0. Questo blocco è stato scritto con B-W, che registra nella posizione 0 il numero 20, puntatore all'ultima posizione scritta nel buffer (nel byte 20 trovi il 13, codice di RETURN). Leggendo con U1 non viene sentito EOF, cioè non viene preso in considerazione il contenuto del byte di posizione 0 come puntatore a fine registrazione, come vedi il buffer viene letto e stampato tutto.

- trasferisce il settore 20,1, scritto con U2, con il comando B-R nel buffer e, senza modificare il puntatore, legge i dati carattere per caratteere con GET# e li stampa, linee da 190 a 205. Dai risultati vedi che la lettura con GET# parte dal carattere di posizione 1, non sente EOF e legge tutto il buffer.

• trasferisce il settore 20,1 con il comando U1 nel buffer e, senza modificare il puntatore, legge i dati carattere per carattere con GET# e li stampa, linee da 210 a 230. Dai risultati vedi che la lettura con GET# parte dal carattere di posizione 0, che in questo caso contiene 0. Scrivendo con U2 non viene posta alcuna segnalazione in posizione 0.

Per leggere con GET#, fino alla segnalazione di EOF (se viene sentita), e stampare 8 dati per riga, si usa la routine in 300.

```
10 REM RANDOMPROVE1
20 REM DIFFERENZE TRA B-R E U1
30 REM DIFFERENZE TRA B-W E U2
33 A$="PRIMO":B$="SECONDO":C$="TERZO"
40 OPEN15,8,15,"I"
45 REM SCRIVE 20,0 CON B-W
50 OPEN2,8,2,"#"
60 PRINT#2,A$:PRINT#2,B$:PRINT#2,C$
70 PRINT#15,"B-W:2,0,20,0"
85 REM SCRIVE 20,1 CON U2
90 OPEN3,8,3,"#"
100 PRINT#3,A$:PRINT#3,B$:PRINT#3,C$
110 PRINT#15,"U2:3,0,20,1"
120 CLOSE2:CLOSE3
123 OPEN4,4:REM APRE STAMPANTE
125 REM LEGGE 20,0 CON B-R
127 PRINT#4,"LEGGE 20,0 CON B-R"
130 OPEN2,8,2,"#"
140 PRINT#15,"B-R:2,0,20,0"
150 GOSUB300:CLOSE2
155 REM LEGGE 20,0 CON U1
157 PRINT#4,"LEGGE 20,0 CON U1"
160 OPEN2,8,2,"#"
170 PRINT#15,"U1:2,0,20,0"
180 GOSUB300:CLOSE2
190 REM LEGGE 20,1 CON B-R
192 PRINT#4,"LEGGE 20,1 CON B-R"
195 OPEN2,8,2,"#"
200 PRINT#15,"B-R:2,0,20,1"
205 GOSUB300:CLOSE2
210 REM LEGGE 20,1 CON U1
```

```
212 PRINT#4,"LEGGE 20,1 CON U1"
215 OPEN2,8,2,"#"
220 PRINT#15,"U1:2,0,20,1"
230 GOSUB300:CLOSE2
250 PRINT#4:CLOSE4:CLOSE15:STOP
300 I=0
305 GET#2,D$:TS=ST:PRINT#4,ASC(D$+CHR$(0));
306 I=I+1:IFI=8THENPRINT#4:I=0
310 IFTS=64THEN PRINT#4:PRINT#4:PRINT#4:RETURN
320 GOTO 305
```

Seguono i risultati, tagliando quelli per i quali viene letto tutto il blocco, dato che, dopo i primi dati significativi, contiene tutti zeri.

```
LEGGE 20,0 CON B-R
 80  82  73  77  79  13  83  69
 67  79  78  68  79  13  84  69
 82  90  79  13

LEGGE 20,0 CON U1
 20  80  82  73  77  79  13  83
 69  67  79  78  68  79  13  84
 69  82  90  79  13  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0
```

```
LEGGE 20,1 CON B-R
 80  82  73  77  79  13  83  69
 67  79  78  68  79  13  84  69
 82  90  79  13  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0
```

```
LEGGE 20,1 CON U1
 0  80  82  73  77  79  13  83
 69  67  79  78  68  79  13  84
 69  82  90  79  13  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0
```

Possiamo concludere che:

- scrivendo con B-W viene registrato sul byte di posizione 0 il puntatore all'ultimo carattere scritto nel buffer,
- scrivendo con U2 il byte di posizione 0 non viene toccato,
- se non si danno comandi per spostare il puntatore, la scrittura dei dati inizia dal byte di posizione 1,
- la lettura con U1 non sente la segnalazione di EOF, e, senza spostare il puntatore, inizia dalla posizione 0,
- la lettura con B-R sente la segnalazione di EOF (ultima posizione raggiunta dal puntatore prima di scrivere), solo se il blocco è stato scritto con B-W, e, senza spostare il puntatore, inizia dalla posizione 1.

Se vuoi, puoi, servendoti del programma di utilità DISPLAY T&S, stampare i settori del dischetto usati nelle prove. Ricordati che il contenuto dei byte viene stampato in esadecimale.

Segue il programma RANDOMPROVE2; esso scrive sui settori 20,2 e 20,3, usando in partenza il puntatore in posizione 2, cioè saltando i primi due byte. Anche qui i due settori vengono scritti con gli stessi dati e con i due comandi disponibili, ma, la

prima volta si scrivono due volte i dati. Poi, nella fase di lettura, vengono letti, ciascuno nei due modi possibili, però, dopo il trasferimento nel buffer, viene posto il puntatore in posizione 0, ottenendo la stampa a partire dal primo byte.

```
10 REM RANDOMPROVE2
20 REM USO DEL PUNTATORE NEL BUFFER A 2
30 REM SCRITTURA CON B-W IN 20,2
31 REM SCRITTURA CON U2 IN 20,3
33 A$="PRIMO":B$="SECONDO":C$="TERZO"
40 OPEN15,8,15,"I"
45 REM SCRIVE 20,2 CON B-W
50 OPEN2,8,2,"#"
55 PRINT#15,"B-P:2,2"
60 PRINT#2,A$:PRINT#2,B$:PRINT#2,C$
65 PRINT#2,A$:PRINT#2,B$:PRINT#2,C$
70 PRINT#15,"B-W:2,0,20,2"
85 REM SCRIVE 20,3 CON U2
90 OPEN3,8,3,"#"
95 PRINT#15,"B-P:3,2"
100 PRINT#3,A$:PRINT#3,B$:PRINT#3,C$
110 PRINT#15,"U2:3,0,20,3"
120 CLOSE2:CLOSE3
123 OPEN4,4:REM APRE STAMPANTE
125 REM LEGGE 20,2 CON B-R
126 REM PUNTATORE A ZERO
127 PRINT#4,"LEGGE 20,2 CON B-R,PUNT.0"
130 OPEN2,8,2,"#"
140 PRINT#15,"B-R:2,0,20,2"
145 PRINT#15,"B-P:2,0"
150 GOSUB300:CLOSE2
155 REM LEGGE 20,2 CON U1
156 REM PUNTATORE A ZERO
157 PRINT#4,"LEGGE 20,2 CON U1,PUNT.0"
160 OPEN2,8,2,"#"
170 PRINT#15,"U1:2,0,20,2"
175 PRINT#15,"B-P:2,0"
180 GOSUB300:CLOSE2
190 REM LEGGE 20,3 CON B-R
191 REM PUNTATORE A ZERO
192 PRINT#4,"LEGGE 20,3 CON B-R,PUNT.0"
195 OPEN2,8,2,"#"
```

```
200 PRINT#15,"B-R:2,0,20,3"
201 PRINT#15,"B-P:2,0"
205 GOSUB300:CLOSE2
210 REM LEGGE 20,3 CON U1
211 REM PUNTATORE A ZERO
212 PRINT#4,"LEGGE 20,3 CON U1,PUNT.0"
215 OPEN2,8,2,"#"
220 PRINT#15,"U1:2,0,20,3"
221 PRINT#15,"B-P:2,0"
230 GOSUB300:CLOSE2
250 PRINT#4:CLOSE4:CLOSE15:STOP
300 I=0
305 GET#2,D$:TS=ST:PRINT#4,ASC(D$+CHR$(0));
306 I=I+1:IFI=8THENPRINT#4:I=0
310 IFTS=64THEN PRINT#4:PRINT#4:PRINT#4:RETURN
320 GOTO 305
```

Riportiamo i risultati, tagliando i blocchi lunghi e pieni di zeri. Abbiamo una differenza nella fase di stampa, rispetto al programma precedente, infatti, muovendo il puntatore in posizione 0 dopo il trasferimento del blocco nel buffer, la prima GET# data sul blocco, scritto con U2 e letto con B-R, interpreta lo zero iniziale come fine file e non stampa dati. Come puoi vedere il puntatore registrato in posizione 0 con il comando B-W, segna per il settore 20,2 il numero 41.

```
LEGGE 20,2 CON B-R,PUNT.0
 41   0   80   82   73   77   79   13
 83   69   67   79   78   68   79   13
 84   69   82   90   79   13   80   82
 73   77   79   13   83   69   67   79
 78   68   79   13   84   69   82   90
 79   13
```

```
LEGGE 20,2 CON U1,PUNT.0
 41   0   80   82   73   77   79   13
 83   69   67   79   78   68   79   13
 84   69   82   90   79   13   80   82
 73   77   79   13   83   69   67   79
 78   68   79   13   84   69   82   90
 79   13   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0
 0   0   0
```

```
LEGGE 20,3 CON B-R,PUNT.0
 0
```

```
LEGGE 20,3 CON U1,PUNT.0
 0   0   80   82   73   77   79   13
 83   69   67   79   78   68   79   13
 84   69   82   90   79   13   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
 0   0   0   0   0   0   0   0
```

Segue il sottoprogramma ALLOCA per dimostrare l'uso corretto del comando B-A.

```
1000 REM ALLOCA
1005 REM TRACCIA TX, SETTORE SX
1010 REM SI SUPPONE APERTO UN CANALE DATI
1015 REM SI SUPPONE APERTO IL CANALE 15
1020 REM SI SUPPONE CHE SX SIA COMPATIBILE CON TX
1025 REM SCARTA LA TRACCIA 18
1030 PRINT#15,"B-A:0";TS;SX
1035 INPUT#15,EN,EM$,ET,ES
1040 IFEN=0THEN RETURN
1045 IFEN<>65THEN STOP
1050 IFET=18THENTX=19:SX=0:GOTO1030
1055 TX=ET:SX=ES:GOTO1030
```

Il sottoprogramma ALLOCA opera così:

- presuppone che sia aperto un canale dati e il canale comandi, che TX e SX contengano il numero di traccia e settore da allocare, che SX sia compatibile con il numero di settori consentiti per TX,
- cerca di allocate TX,SX, linea 1030,
- legge le quattro variabili di controllo sul canale 15, linea 1035,
- se EN=0 esce, e il settore TX,SX è stato allocato, linea 1040,
- se EN=65, significa che il settore richiesto è già occupato, allora controlla che la nuova traccia suggerita in ET non sia la 18 (directory), e, in caso cambia traccia e settore; altrimenti pone in TX e SX la traccia e il settore suggeriti in ET ed ES e va ad allocare il nuovo settore, linee 1050 e 1055,
- se si verifica un errore di altro tipo si ha uno STOP, linea 1045,
- in realtà si dovrebbe aggiungere la linea:

1043 IF EN=65 AND ET=0 THEN STOP

e questo significherebbe che viene controllato se il dischetto è pieno.

Segue il programma RANDOMPROVE3, per dimostrare l'uso di B-W e di B-R, leggendo dal byte di posizione 0 la posizione raggiunta dal puntatore, spostandolo con B-P, e continuando a registrare dati nel buffer.

```
10 REM RANDOMPROVE3
15 REM USO DELLA POSIZIONE RAGGIUNTA
20 REM DAL PUNTATORE NEL BUFFER
30 REM SCRITTURA CON B-W, LETTURA CON B-R
31 REM AGGIUNTA DATI SPOSTANDO IL PUNTATORE
32 REM IN PASE AL CONTENUTO DELLA POSIZIONE 0
33 A$="PRIMO":B$="SECONDO"
40 OPEN15,8,15,"I"
45 REM SCRIVE 20,4 CON B-W PRIMO DATO
50 OPEN2,8,2,"#"
55 PRINT#15,"B-P:2,2"
60 PRINT#2,A$
70 PRINT#15,"B-W:2,0,20,4"
80 CLOSE2
83 OPEN4,4:REM APRE STAMPANTE
84 PRINT#4,"DOPO SCRITTURA PRIMO DATO"
85 REM LEGGE 20,3 CON B-R
87 REM SPOSTANDO IL PUNTATORE IN POSIZIONE 0
```

```
90 OPEN2,8,2,"#"
93 PRINT#15,"B-R:2,0,20,4"
94 PRINT#15,"B-P:2,0":GET#2,P$
95 X=ASC(P$+CHR$(0)):X=X+1
96 PRINT#4,"PUNTATORE: ";X-1
99 GOSUB300
100 REM SPOSTA IL PUNTATORE AL VALORE LETTO- IN P$
105 REM PIU' 1, PER SCRIVERE SECONDO DATO
110 PRINT#15,"B-P:2";X
115 PRINT#2,B$
120 PRINT#15,"B-W:2,0,20,4"
125 CLOSE2
130 REM LEGGE 20,3 CON B-R
135 PRINT#4,"DOPO SCRITTURA SECONDO DATO"
140 OPEN2,8,2,"#"
145 PRINT#15,"B-R:2,0,20,4"
147 PRINT#15,"B-P:2,0"
150 GOSUB300:CLOSE2
250 PRINT#4:CLOSE4:CLOSE15:STOP
300 I=0
305 GET#2,D$:TS=ST:PRINT#4,ASC(D$+CHR$(0));
306 I=I+1:IFI=8THENPRINT#4:I=0
310 IFTS=64THEN PRINT#4:PRINT#4:PRINT#4:RETURN
320 GOTO 305
```

Seguono i risultati di RANDOMPROVE3.

```
DOPO SCRITTURA PRIMO DATO
PUNTATORE:  7
 0  80  82  73  77  79  13

DOPO SCRITTURA SECONDO DATO
 15  0  80  82  73  77  79  13
 83  69  67  79  78  68  79  13
```

Come vedi, dopo aver scritto PRIMO, il puntatore indica 7, lo spostiamo alla posizione 8 e scriviamo SECONDO; alla fine il byte di posizione 0 reca 15.

Riepiloghiamo le operazioni da eseguire per scrivere un record su un file random:

- Aprire il canale 15.
- Aprire il canale per inviare i dati (OPEN..."#").
- Se si deve aggiornare o se il record logico nuovo non occupa un intero settore, leggere con B-R o U1, il blocco fisico a cui appartiene il record logico.
- Provvedere a spostare, o meno, il puntatore nel buffer.
- Scrivere con PRINT# nel buffer, usando lo stesso lfn usato nella OPEN..."#". Dopo ogni trasferimento di dato, il puntatore avanza alla posizione seguente l'ultimo carattere scritto (di solito il CHR$(13)).
- Scrivere il buffer sul dischetto con B-W o U2.
- Chiudere il canale dati e il canale comandi.

Riepiloghiamo le operazioni da fare per leggere un record da un file random:

- Aprire il canale 15.
- Aprire un canale per i dati con OPEN..."#".
- Leggere il blocco fisico a cui appartiene il record logico, con B-R o U1.
- Spostare il puntatore, se necessario.
- Leggere con INPUT# o con GET#, con lo stesso lfn usato nella OPEN del canale dei dati.
- Chiudere il canale dei dati e il canale 15.

Vediamo ora i comandi che consentono una programmazione più sofisticata; li raccomandiamo solo agli appassionati programmatori.

BLOCK-EXECUTE, abbreviazione facoltativa B-E

Consente di caricare in un buffer dell'unità 1541 il contenuto di un settore del dischetto, e, considerandolo un programma in linguaggio macchina, lo manda in esecuzione a partire dal byte di posizione 0 del buffer. Per un corretto funzionamento del comando, devono essere verificate le seguenti condizioni:

- il programma in linguaggio macchina deve avere la sua prima istruzione nel primo byte del settore,
- esso deve occupare al massimo 256 byte,
- il settore del dischetto deve essere stato preventivamente scritto con il comando U2, per non danneggiare il byte in posizione 0,
- il programma deve terminare logicamente con l'istruzione RTS (ReTurn from Subroutine).

Memory-Write, abbreviazione obbligatoria M-W

Consente di registrare, al massimo 34 byte, nella RAM dell'unità 1541. I parametri del comando sono in ordine: byte basso e byte alto dell'indirizzo di memoria da

dove iniziare a memorizzare, numero dei caratteri, byte da trasferire. Questi parametri devono essere passati con la funzione CHR$, avente per argomento il numero decimale che rappresenta il dato. Per esempio, se vogliamo scrivere all'indirizzo 1794, dobbiamo calcolare:

1794/256 dà come parte intera 7, byte alto
1794-7*256=1794-1792=2, byte basso,

in conseguenza i primi due parametri sono: CHR$(2)CHR$(7).

Se vogliamo trasferire 10 byte, il terzo parametro sarà: CHR$(10).

Dopo aver scritto i 10 byte del codice macchina dobbiamo prenderne il valore decimale e usarlo come argomento dei 10 CHR$ successivi.

Il comando si limita a memorizzare il codice a partire da un certo indirizzo. Per mandare il codice in esecuzione dobbiamo usare il comando M-E.

Memory-Read, abbreviazione obbligatoria M-R

Consente di leggere, tramite il canale comandi, il contenuto di un byte della memoria RAM o ROM dell'unità 1541 (possiamo considerarla equivalente alla funzione PEEK per il calcolatore). I parametri da passare sono il byte basso e il byte alto dell'indirizzo di memoria da cui vogliamo cominciare a leggere, passati con la funzione CHR$; così:

PRINT # 15,"M-R:"CHR$(x)CHR$(y)

dove y*256+x è l'indirizzo scelto per leggere.
Il comando pone sul canale 15 il contenuto del byte, di cui si è passato l'indirizzo; per leggere il contenuto si deve eseguire:

GET#15,C$

e in C$ si ha il dato. L'indirizzo viene incrementato automaticamente.

Segue il programma LETTURA1541, come esempio.

```
10 REM LETTURA1541
20 REM LETTURA MEMORIA
30 REM VIENE CHIESTO INDIRIZZO DI PARTENZA
40 REM VIENE CHIESTO NUMERO BYTE DA LEGGERE
50 OPEN15,8,15
60 INPUT"   INDIRIZZO: ";I
70 INPUT"   NUMERO BYTE: ";N
80 I1=INT(I/256):I2=I-I1*256
90 PRINT#15,"M-R:"CHR$(I2)CHR$(I1)
100 OPEN4,4:PRINT#4,"LETTURA DA: ";I
110 L=0:FORK=1TON
120 GET#15,C$
130 PRINT#4,ASC(C$+CHR$(0));
140 L=L+1:IFL=8THENPRINT#4:L=0
150 NEXTK:PRINT#4:CLOSE4:CLOSE15
160 STOP
```

Con esso puoi leggere N byte a partire da un indirizzo I.

Memory-Execute, abbreviazione obbligatoria M-E

Consente di mandare in esecuzione un programma in linguaggio macchina che si trovi memorizzato o nella ROM o nella RAM dell'unità 1541.

In particolare, questo comando deve essere usato per mandare in esecuzione i programmi memorizzati con il comando M-W. I parametri da passare con CHR$, sono il byte basso e il byte alto dell'indirizzo del primo byte del programma da eseguire.

Esistono, inoltre, i comandi di tipo USER; essi, esclusi i primi due, che abbiamo già esaminato, servono per operare dei salti a particolari indirizzi della memoria dell'unità1541. Gli indirizzi sono rilevabili dalla Tabella 3.4, prima riportata.
Tu puoi memorizzare a partire da questi indirizzi (quelli situati in RAM) le tue routine in linguaggio macchina, e poi mandarle in esecuzione con Ui, con la solita PRINT#15. I comandi da U3 a U8 fanno saltare all'interno di uno dei buffer dell'unità.

Con gli ultimi comandi visti si possono programmare operazioni che modificano i contenuti dei buffer, e, in conseguenza, dei file, e vengono eseguite all'interno dell'unità 1541.

3.2 ESEMPIO DI ARCHIVIO RANDOM

Riportiamo un esempio di archivio RANDOM con indice. Il programma ARCHI-RANDOM fa le seguenti cose:

- crea su dischetto un archivio RANDOM con indice primario sequenziale,
- lista tutto l'archivio in ordine secondo l'indice primario (ordinato in senso crescente),
- aggiorna l'archivio:
  - aggiungendo record,
  - modificando record esistenti,
  - cancellando record esistenti,
- crea un indice secondario in base a un qualunque campo del record, escluso il primo,
- lista tutto l'archivio in ordine secondo l'indice secondario (ordinato in senso decrescente).

Il record logico è stato strutturato in 14 campi (NC=14), di tipo stringa:

1) Cognome	20 car.
2) Nome	15 "
3) Indirizzo	30 "
4) Città (compr. CAP)	25 "
5) Provincia	11 "
6) Telefono	10 "
7) Luogo nascita	20 "
8) Data nascita	8 "
9) Titolo studio	20 "
10) Occupazione attuale	20 "
11) Occupazione prec.	20 "
12) Stato civile	1 "
13) Nota 1	20 "
14) Nota 2	20 "

nel computo dei caratteri del record si deve aggiungere ad ogni campo un carattere di fine campo (noi usiamo il RETURN, CHR$(13)). In conseguenza il record logico è formato da:

21+16+31+26+12+11+21+9+21+21+21+2+21+21=254

caratteri. I campi sono tutti di lunghezza fissa; i primi due caratteri di ogni settore non sono utilizzati, il record logico coincide con il blocco fisico.

Noi abbiamo strutturato i campi per contenere il solito archivio Anagrafico, ma non è difficile cambiare il numero dei campi del record, le descrizioni dei campi, le lunghezze in caratteri di ogni campo; infatti tutte queste definizioni sono raggruppate all'inizio del programma e sono facilmente modificabili. Inoltre noi abbiamo considerato una chiave di ordinamento formata dai primi due campi di ogni record: Cognome e Nome. In base a questa chiave viene preparato l'indice primario; esso contiene il cognome, il nome e l'indirizzo, traccia e settore, dove si trova il record corrispondente del file principale.

Nella fase di creazione del file, viene preparato anche l'indice principale; esso viene costruito in memoria in 3 vettori opportunamente dimensionati, uno per la chiave, cognome + nome, uno per la traccia e l'altro per il settore. I record possono essere caricati anche non in ordine di chiave crescente, infatti il programma provvede a ordinare l'indice prima di scriverlo su disco. Il programma accetta anche cognomi e nomi uguali, e, in fase di ricerca chiede quale occorrenza della chiave si vuole (1, 2,...). Il file indice è di tipo sequenziale, formato da record di 3 campi: chiave, traccia e settore. L'indice principale viene caricato in memoria all'inizio dell'elaborazione (salvo che nella fase di creazione del file), può venire modificato, e viene riscritto su disco quando si termina l'elaborazione. L'indice principale, di nome INDI1, viene mantenuto in ordine di chiave crescente.

Inoltre si può costruire un indice secondario, di nome INDI2, scegliendo come campo chiave un campo qualunque del record, salvo il primo. Tale indice viene ordinato in senso decrescente e memorizzato su disco. Si può ottenere una lista dell'archivio secondo INDI2. L'indice secondario può essere rifatto tutte le volte che si vuole, cancellando il precedente.

Il programma usa un dischetto apposito solo per i dati; esso registra sulla traccia 1, settore 0, i dati generali del file, e cioè: nome dell'archivio, che coincide con il nome del disco, data di aggiornamento, numero record presenti, indirizzo di traccia e settore dell'ultimo record registrato. I record sono registrati a partire dalla traccia 1, settore 1.
Nella directory del dischetto dati sono registrati solo i file indice, INDI1 e INDI2, che sono dei normali file sequenziali. Il file random non compare nella directory; per questa ragione sul dischetto dati non si deve mai fare l'operazione VALIDATE. Infatti tale operazione cancella tutti i settori occupati, ma non registrati nella directory e ne rende disponibile lo spazio.

Il problema dell'aggiornamento del file risulta molto più semplice che nel caso del file sequenziale, dato che si può accedere a qualunque record, dopo averne trovato l'indirizzo servendosi del file indice. Ti sembrerà ridondante aver scritto il cognome e il nome sia nel record principale che nel file indice INDI1; ma abbiamo preferito avere un record completo nel file principale, cosa, del resto, necessaria per poter creare indici secondari.

Segue il listato del programma; come puoi osservare abbiamo cercato di impaginare il programma in modo che sia abbastanza facile trovare i diversi blocchi. Inoltre, il programma è stato scritto in modo modulare; questo rende possibile operare le variazioni senza troppi problemi. Ti consigliamo di adattare il programma alle tue esigenze. L'unico blocco che è molto legato alla struttura del nostro record è quello relativo alla stampa, ma è semplice da modificare.
Devi, però, ricordare che il programma si basa su un record logico coincidente con il settore, e che la somma delle lunghezze dei campi, compreso il carattere di fine campo, non deve superare 254.

```
10 REM ARCHIRANDOM
15 REM-----------------------------------
20 REM DEFINIZIONE COSTANTI E VARIABILI
25 REM-----------------------------------
30 NC=14:REM **NUMERO CAMPI RECORD**
35 REM       **********************
40 S1$="DATA ULTIMO AGG.       "
45 S2$="FINITO SPAZIO ASSEGNATO"
50 SP$="                                  "
55 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(99)+CHR$(99)
60 DIMD$(NC):DIMY$(NC):DIML(NC)
65 DATA"COGNOME:","NOME    :","INDIR.  :"
70 DATA"CITTA' :","PROV.   :","TELEF.  :"
75 DATA"L.NASC.:","D.NASC.:","T.STUD.:"
80 DATA"OCC.AT.:","OCC.PR.:","ST.CIV.:"
85 DATA"NOTA 1 :","NOTA 2 :"
90 DATA20,15,30,25,11,10,20,8,20,20,20,1,20,20
95 FORK=1TONC:READD$(K):NEXTK
100 FORK=1TONC:READL(K):NEXTK
105 REM **PRESENTAZIONE MENU' E SCELTE**
110 REM ******************************
115 OPEN15,8,15
120 PRINT"□□□";TAB(10);"GESTIONE ARCHIVIO":PRINT
125 PRINTTAB(10)"1=INIZIO EX-NOVO"
130 PRINTTAB(10)"2=AGGIORNAMENTO"
135 PRINTTAB(10)"3=LISTA PRINCIPALE"
140 PRINTTAB(10)"4=CREAZ.IND.SEC."
145 PRINTTAB(10)"5=LISTA SECONDARIA"
150 PRINTTAB(10)"9=FINE"
155 INPUT"COSA SCEGLI ";X
160 IFX<1ORX>5ANDX<>9THENPRINT"□";:GOTO155
```

```
165 PRINT:GOSUB580
170 R$="":PRINTTAB(10)"MONTA DISCO DATI"
175 GETR$:IFR$=""THEN175
180 IFX=1THEN205
185 PRINT#15,"I"
190 GOSUB480:N=K:PRINT"PRESENTI ";K;" RECORD"
195 PRINTS1$;G1$;"/";M1$;"/";A1$
200 IFX<>2THEN225
205 REM-------------------------
210 REM INIZIO EX-NOVO ARCHIVIO
215 REM-------------------------
220 PRINT"     QUANTI RECORD ";:INPUT N
225 DIMC$(N),T%(N),S%(N)
230 IFX<>1THENGOSUB520
235 IFX=9THEN1290
240 ONXGOTO635,770,1095,1180,1240
245 STOP
250 REM **INGRESSO DATI**
255 REM *****************
260 PRINT"INGRESSO DATI"
265 FORJ=1TONC:Y$(J)="":NEXTJ
270 PRINT"PER USCIRE $ PER COGNOME";
275 PRINT"SE MANCANO DATIPREMI SOLO RETURN"
280 PRINTD$(1);:INPUTY$(1)
285 IFY$(1)="$"THENW=1:RETURN
290 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
295 REM **SISTEMA LUNGHEZZA DATI**
300 REM ***************************
305 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
310 NEXTJ:GOSUB960:RETURN
315 REM **SCRITTURA INDICE PRINCIPALE**
320 REM *******************************
325 PRINT#15,"S0:INDI1":PRINT#15,"I"
330 OPEN10,8,10,"INDI1,S,W":GOSUB550
335 FORJ=1TOK
340 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
345 GOSUB550:NEXTJ
350 CLOSE10:RETURN
355 REM **SCRITTURA NEL BUFFER**
360 REM ************************
365 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
```

```
370 PRINT#11,Y$(J);CH$;:GOSUB550
375 NEXTJ:RETURN
380 REM **ALLOCA TRACCIA E SETTORE**
385 REM ****************************
390 PRINT#15,"B-A:"0;T;S
395 INPUT#15,EN,EM$,ET,ES
400 IFEN=0THENRETURN
405 IFEN<>65THEN570
410 IFET=18THENT=19:S=0:GOTO390
415 T=ET:S=ES:GOTO390
420 REM **PUNTATORE NEL BUFFER**
425 REM ************************
430 PRINT#15,"B-P:"11;2:GOSUB550:RETURN
435 REM **SCRITTURA RECORD**
440 REM ********************
445 PRINT#15,"U2:"11;0;T;S:GOSUB550:RETURN
450 REM **LETTURA RECORD**
455 REM ******************
460 PRINT#15,"I":OPEN11,8,11,"#":GOSUB550
465 PRINT#15,"U1:"11;0;T;S:GOSUB550:GOSUB430
470 FORJ=1TONC:INPUT#11,Y$(J):GOSUB550
475 NEXTJ:CLOSE11:RETURN
480 REM **LETT.DATI DISCO**
485 REM ********************
490 OPEN11,8,11,"#":GOSUB550
495 PRINT#15,"U1:"11;0;1;0:GOSUB550
500 GOSUB420:INPUT#11,R$,K,T,S:GOSUB550
505 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
510 A1$=RIGHT$(R$,2):CLOSE11
515 TT=T:SS=S:RETURN
520 REM **LETT. INDICE**
525 REM ****************
530 OPEN10,8,10,"INDI1,S,R":GOSUB550
535 FORJ=1TOK
540 INPUT#10,C$(J),T%(J),S%(J):GOSUB550
545 NEXTJ:CLOSE10:RETURN
550 REM **ROUTINE ERRORE**
555 REM ******************
560 INPUT#15,EN,EM$,ET,ES
565 IFEN=0THENRETURN
570 PRINT"ERRORE DISCO"
575 PRINTEN,EM$,ET,ES:CLOSE15:STOP
```

```
580 REM **DATA DISCO**
581 REM **************
585 PRINT"DATA PER DISCO"
590 INPUT"  GG,MM,AA[redacted]";G$,M$,A$:RETURN
595 REM **SCRITT. IND. SEC.**
596 REM *********************
600 PRINT#15,"S:INDI2":PRINT#15,"I"
605 OPEN10,8,10,"INDI2,S,W"
610 FORJ=1TOK
615 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
620 GOSUB550:NEXTJ:CLOSE10:RETURN
625 GETR$:IFR$=""THEN625
630 RETURN
635 REM-------------------------
640 REM INIZIALIZZAZIONE DISCO
645 REM-------------------------
650 PRINT"NOME DISCO";:INPUTN$:T=1:S=0
655 REM **DATI DISCO SETTORE 1,0**
660 REM ***************************
665 PRINT#15,"N0:"+N$+",99"
670 CLOSE15:OPEN15,8,15:PRINT#15,"I"
675 REM **DATA AGG. E NUM. RECORD**
680 REM ****************************
685 OPEN11,8,11,"#":GOSUB550:GOSUB390:GOSUB430
690 K=0
695 PRINT#11,G$M$A$CH$;K;CH$;1;CH$;1;CH$;
700 REM **PRIMO SETTORE DATI 1,1**
705 REM ***************************
710 GOSUB550
715 PRINT#15,"U2:"11;0;T;S:GOSUB550
720 K=1:W=0:T=1:S=1
725 GOSUB250
730 IFW=1THENK=K-1:CLOSE11:GOTO1290
735 GOSUB380:GOSUB420:GOSUB355
740 REM **AGGIORNAMENTO INDICE**
745 REM ************************
750 C$(K)=Y$(1)+Y$(2):T%(K)=T:S%(K)=S
755 GOSUB435:K=K+1
760 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO1290
765 GOTO725
770 REM---------------
775 REM AGGIORNAMENTO
```

```
780 REM---------------
785 PRINTTAB(10);"□□AGGIORNAMENTO ARCHIVIO"
790 PRINT:PRINTTAB(10);"1=CORREZIONE"
795 PRINTTAB(10);"2=AGGIUNTA ELEM."
800 PRINTTAB(10);"3=CANCELL.ELEM."
805 PRINTTAB(10);"9=FINE"
810 INPUT"COSA SCEGLI ";X:IFX=9THEN1290
815 IFX<1ORX>3THEN785
820 IFX=2THEN1010
825 IFX=3THEN1050
830 GOTO915
835 REM **RICERCA RECORD**
840 REM ******************
845 PRINT"□";D$(1);:INPUTY$(1):W=0
850 IFY$(1)="*"THENW=1:RETURN
855 PRINTD$(2);:INPUTY$(2)
860 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
865 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
870 INPUT"QUALE OCCORRENZA ";X
875 IFX<=0THENPRINT"□";:GOTO870
880 FORJ=1TOK:IFC$(J)=Y$(1)+Y$(2)THEN900
885 NEXTJ
890 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
895 GOSUB625:GOTO835
900 IFX<>1THENX=X-1:GOTO885
905 T=T%(J):S=S%(J)
910 I=J:J=K:NEXTJ:RETURN
915 REM **CORREZIONE**
920 REM ***************
925 GOSUB835:IFW=1THEN785
930 GOSUB450
935 REM **VA A MODIFICA DATI**
936 REM **********************
940 GOSUB960
945 PRINT#15,"I":OPEN11,8,11,"#":GOSUB420:GOSUB355
950 C$(I)=Y$(1)+Y$(2):T%(I)=T:S%(I)=S
955 GOSUB435:CLOSE11:GOTO830
960 REM **CONTROLLO DATI E MODIFICHE**
965 REM ******************************
970 PRINT"□CONTROLLO DATI E MODIFICHE"
975 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
980 INPUT"■TUTTO BENE S/N ";X$:IFX$="S"THENRETURN
```

```
985 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
990 IFX=0THENRETURN
995 IFX>NCORX<1THENPRINT"□":GOTO980
1000 Y$(X)=""
1003 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
1005 GOTO970
1010 REM **AGGIUNTA**
1015 REM ************
1020 OPEN11,8,11,"#":GOSUB550
1025 PRINT"□AGGIUNTA NUOVI RECORD"
1030 PRINT"PRESENTI ";K;" RECORD"
1035 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
1040 GOSUB625
1045 T=TT:S=SS:W=0:K=K+1:GOTO725
1050 REM **CANCELLAZIONE**
1055 REM ******************
1060 M=0
1065 GOSUB835:IFW=1THEN1085
1070 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
1075 M=M+1
1080 PRINT#15,"B-F:"0;T;S:GOSUB550:GOTO1065
1085 REM SIST. INDICE
1090 GOSUB1355:K=K-M:GOTO1315
1095 REM-------------
1100 REM LISTA FILE
1105 REM-------------
1110 T$="LISTA PER INDICE PRIM."
1115 PRINT"ACCENDI STAMPANTE     PREMI UN TASTO"
1120 GETR$:IFR$=""THEN1120
1125 PRINT"□";T$:OPEN4,4
1130 PRINT#4:PRINT#4:PRINT#4,T$
1135 FORJ=1TO10:PRINT#4:NEXTJ
1140 L=2:FORI=1TOK:T=T%(I):S=S%(I):GOSUB450
1145 PRINT#4,Y$(1);"  ";Y$(2)
1150 PRINT#4,Y$(3);"  ";Y$(4);"  ";Y$(5)
1155 FORM=6TONC:PRINT#4,D$(M);Y$(M):NEXTM
1160 PRINT#4:PRINT#4
1165 IFL=5THENPRINT#4:L=0
1170 L=L+1:NEXTI
1175 CLOSE4:GOTO1340
1180 REM-------------------------------
1185 REM CREAZIONE INDICE SECONDARIO
```

```
1190 REM--------------------------------
1195 PRINT"□INDICE SECONDARIO"
1200 INPUT"□QUALE CAMPO ";X$
1205 X=VAL(X$):IFX<2ORX>NCTHENGOTO1200
1210 GOSUB315
1215 FORI=1TOK:T=T%(I):S=S%(I):GOSUB450
1216 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
1220 C$(I)=Y$(X):NEXTI
1225 GOSUB595:GOSUB520
1230 PRINT"FINITO IND. SEC."
1235 GOTO1340
1240 REM--------------------------------
1245 REM LISTA PER INDICE SECONDARIO
1250 REM--------------------------------
1255 PRINT"□LISTA IND. SEC."
1260 OPEN10,8,10,"INDI2,S,R":GOSUB550
1265 FORJ=1TOK:INPUT#10,C$(J),T%(J),S%(J)
1270 GOSUB550:NEXTJ
1275 CLOSE10:GOSUB1425:GOSUB595
1280 T$="LISTA IND. SEC. "
1285 GOTO1115
1290 REM **CHIUSURA**
1295 REM ************
1300 PRINT"CHIUSURA ARCHIVIO"
1305 PRINT"IND. PRINC. DA ORDIN. S/N ":INPUTR$
1310 IFR$="S"THENGOSUB1355
1315 GOSUB315:OPEN11,8,11,"#":GOSUB550
1320 PRINT#15,"B-P:"11,2
1325 PRINT#11,G$M$A$CH$;K;CH$;T;CH$;S;CH$;
1330 GOSUB550
1335 PRINT#15,"U2:"11;0;1;0:GOSUB550:CLOSE11
1340 PRINT"FINITO AGGIORNAMENTO"
1345 PRINT"SONO PRESENTI ";K;" RECORD"
1350 CLOSE15:STOP
1355 REM **ORDINAMENTO CRESCENTE**
1360 REM **************************
1365 L=K-1
1370 W=0
1375 FORJ=1TOL
1380 IFC$(J)<=C$(J+1)THEN1405
1385 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
1390 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
1395 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
```

```
1400 W=1
1405 NEXTJ
1410 IFW=0THENRETURN
1415 IFL=1THENRETURN
1420 L=L-1:GOTO1370
1425 REM **ORDINAMENTO DECRESCENTE**
1430 REM ****************************
1435 L=K-1
1440 W=0
1445 FORJ=1TOL
1450 IFC$(J)>=C$(J+1)THEN1475
1455 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
1460 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
1465 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
1470 W=1
1475 NEXTJ
1480 IFW=0THENRETURN
1485 IFL=1THENRETURN
1490 L=L-1:GOTO1440
```

Il programma è formato dai seguenti blocchi:

DEFINIZIONE COSTANTI E VARIABILI
linee 15-100
sono definite alcune costanti, i vettori per contenere le descrizioni dei campi del record, le lunghezze dei campi e i dati. Si usa l'istruzione READ per riempire i vettori di costanti. Volendo modificare la struttura del record logico, si deve lavorare in questo blocco.

PRESENTAZIONE MENU' E SCELTE
linee 105-200
viene presentato il menù:
1=INIZIO EX-NOVO
2=AGGIORNAMENTO
3=LISTA PRINCIPALE
4=CREAZ.IND.SEC.
5=LISTA SECONDARIA
9=FINE
e, controllata la scelta, viene chiesto di montare il disco dati; se non è stata scelta la fase di creazione, viene segnalato quanti record sono presenti nell'archivio.

INIZIO EX-NOVO ARCHIVIO
linee 205-245 e 635-765
viene chiesto il numero di record da trattare, vengono dimensionati i tre vettori

C$(N), T%(N) e S%(N), per l'indice principale. Da questo blocco si passa sempre, qualunque sia stata la scelta. Dopo le prime operazioni il programma segue strade diverse a seconda della scelta iniziale. Se la scelta è 1, si ha:

- Formattazione del disco, con memorizzazione del nome dell'archivio (che coincide con il nome del disco), della data e dell'indirizzo iniziale del record dati (traccia 1, settore 1) nel settore 0 della traccia 1. Tale settore viene sempre aggiornato, prima della chiusura, se non altro per la data, e vi viene registrato anche il numero di record presenti nell'archivio e l'indirizzo dell'ultimo settore usato.
- Viene eseguito il sottoprogramma di richiesta dati (linea 250), e, quando i dati sono terminati, viene scritto su disco il file indice INDI1, aggiornato il settore 1,0 e il programma termina.

AGGIORNAMENTO

linee 770-1090
viene presentato il menù:

1=CORREZIONE
2=AGGIUNTA ELEM.
3=CANCELL.ELEM.
9=FINE.

Dopo il controllo della scelta, il programma va a:

- Correzione, linee 915-955

servendosi dei relativi sottoprogrammi viene chiesto il record da modificare, viene modificato, viene riscritto e viene aggiornato l'indice in memoria. Il programma consente di modificare anche i due campi chiave; ovviamente, se si modifica la chiave, è necessario riordinare l'indice.

- Aggiunta, linee 1010-1045

servendosi dei relativi sottoprogrammi, vengono aggiunti record e aggiornato l'indice. Prima di chiudere deve essere riordinato l'indice.

- Cancellazione, linee 1050-1090

i record vengono cancellati ponendo nella chiave una stringa che inizia con un codice ASCII, ripetuto 3 volte, superiore al codice delle lettere dell'alfabeto. In questo modo, ordinando l'indice in modo crescente, i record cancellati vanno a finire in fondo e poi non vengono riscritti su disco. I settori che non servono più vengono liberati con la funzione B-F.

LISTA FILE

linee 1095-1175
viene listato il file secondo un tracciato che occupa 11 linee, rispettando il cambio di foglio (66 linee). Riportiamo la lista di un record.

```
CARLINI                    GIULIO
VIALE ROSE 45                         20100 MILANO            MI
TELEF.  :12345678
L.NASC. :CORSICO
D.NASC. :15061940
T.STUD. :PERITO CHIMICO
OCC.AT. :ANALISTA LABORATORIO
OCC.PR. :TECNICO LABORATORIO
ST.CIV. :C
NOTA 1  :OTTIMO ELEMENTO
NOTA 2  :MOLTO SCRUPOLOSO
```

Questo blocco di programma dovrà probabilmente essere modificato, se si passa ad un archivio di altra natura e diversamente strutturato.

CREAZIONE INDICE SECONDARIO
linee 1180-1235
dopo aver chiesto il numero del campo, da usare come chiave di ordinamento, per creare l'indice secondario (supponendo che possa variare tra 2 e NC), si crea l'indice secondario servendosi in memoria degli stessi vettori usati per l'indice principale. Alla fine l'indice secondario viene scritto sul dischetto con il nome INDI2, cancellando, se esiste, un precedente indice secondario.

LISTA PER INDICE SECONDARIO
linee 1240-1285
viene usato INDI2 come indice per listare il file.

CHIUSURA
linee 1290-1350
viene chiesto se l'indice principale è da ordinare; se la risposta è S, esso viene ordinato, e, comunque, viene riscritto su disco. Viene aggiornato il settore 1,0 per i dati generali del disco che sono:

- data di aggiornamento,
- numero record presenti,
- indirizzo traccia e settore ultimo settore riempito (nell'allocazione di settori per aggiungere record si parte da questo indirizzo).

Il programma termina segnalando sul video il numero dei record presenti nell'archivio.

ORDINAMENTO CRESCENTE
linee 1355-1420
è la routine di ordinamento a bolle. Essa ordina in senso crescente il vettore C$(N), trascinandosi dietro nell'ordinamento i due vettori T%(N) e S%(N). Se il numero dei record è abbastanza elevato, tale routine risulta piuttosto lenta (vedi gli esempi di ordinamento nel primo volume), ma essa può essere facilmente sostituita con una più efficiente.

ORDINAMENTO DECRESCENTE
linee 1425-1490
è la routine di ordinamento a bolle, ma ordina in senso decrescente. Vale quanto detto per la routine precedente.

INGRESSO DATI
linee 250-310
richiede i dati per un record, rende i campi della lunghezza fissa stabilita, chiede conferma dei dati ricevuti e consente di modificarli. Per uscire dalla routine senza fornire dati, basta rispondere con il carattere "$" alla richiesta del cognome. Se il record è stato ricevuto la routine ritorna lo switch W=0, mentre se il record non è stato ricevuto ritorna lo switch W=1.
La lunghezza dei campi (linee 295-310) è importante, infatti i record devono mantenere i campi della lunghezza stabilita. Qualora si desideri accedere a un particolare campo nel record, si deve poter porre il puntatore al valore previsto; se i campi non sono stati scritti come si deve, si rischiano grossi pasticci.

SCRITTURA INDICE PRINCIPALE
linee 315-350
va a scrivere il file sequenziale INDI1 su disco.

SCRITTURA NEL BUFFER
linee 355-375
trasferisce il vettore dei dati nel buffer.

ALLOCA TRACCIA E SETTORE
linee 380-415
alloca un settore, sistemando la BAM.
Non viene controllato se EN=65 e ET=0; si suppone di lavorare con un numero di record compatibile con la capacità del dischetto.

PUNTATORE NEL BUFFER
linee 420-430
pone il puntatore nella posizione 2 del buffer.

SCRITTURA RECORD
linee 435-445
scrive il buffer su disco nel settore T,S.

LETTURA RECORD
linee 450-475
legge il settore T,S nel buffer e poi trasferisce i campi nel vettore Y$(NC).

LETTURA DATI DISCO
linee 480-515
legge i dati generali del disco.

LETTURA INDICE
linee 520-545
legge in memoria INDI1. Non viene controllato l'EOF, dato che si legge un numero fisso di record.

ROUTINE ERRORE
linee 550-575
controlla gli errori relativi alle operazioni disco.

DATA DISCO
linee 580-590
richiede la data di aggiornamento.

SCRITTURA INDICE SECONDARIO
linee 595-630
scrive su disco INDI2.

RICERCA RECORD
linee 85-910
ricerca un determinato record, dopo aver chiesto: cognome, nome e occorrenza (sono ammessi record multipli). Segnala se non lo trova.

Elenco variabili e costanti

NC=14, numero campi del record
S1$, stringa descrittiva
S2$, stringa descrittiva
SP$,30 spazi, serve per aggiustare le lunghezze dei campi, deve essere lunga come il campo più lungo
CH$=CHR$(13), carattere RETURN, usato come fine campo
LIM$=CHR$(99)+CHR$(99)+CHR$(99), serve per la chiave dei record cancellati

D$(NC), descrizioni campi record
Y$(NC), campi dati
L(NC), lunghezze dei campi dati
K,J,L variabili di controllo
N, numero record
R$, variabile per risposta
X, variabile di comodo
G1$,M1$,A1$, data precedente
C$(N), vettore per chiave indice
T%(N), vettore per numero traccia
S%(N), vettore per numero settore
EN,EM$,ET,ES, variabili per routine errore
T,S, indirizzo traccia e settore
G$,M$,A$, data aggiornamento
N$, nome disco e archivio
W, switch per lettura dati
X$, variabile per risposte
TT,SS, indirizzo settore precedente
M, numero record cancellati

Il programma a volte invia dei messaggi sul video e non chiede risposta, per proseguire devi premere un tasto. Se viene chiesta la risposta affermativa, devi premere S.

Le operazioni disco vengono svolte usando i comandi U1 e U2.

Vediamo ora le critiche al programma:

- non abbiamo dedicato particolare cura ai quadri video,
- non abbiamo dedicato particolare cura alla stesura dei listati,
- non abbiamo programmato liste parziali dell'archivio,
- il programma dopo ogni funzione termina, per proseguire devi dare ancora RUN,
- SUL DISCHETTO DEI DATI NON PUO' ESSERE ESEGUITA LA FUNZIONE VALIDATE, PENA LA PERDITA DEI DATI.

Prima di chiudere, facciamo un pò di conti, cioè vediamo quali possono essere le dimensioni di un archivio gestito dal programma ARCHIRANDOM.

Il programma occupa, prima di dare RUN, 7397 byte, quindi, restano liberi per le variabili 38911-7397=31514 byte.
Il dischetto ha 664 settori disponibili. Un settore è occupato per i dati generali del disco. Per ogni record vengono occupati:

- 1 settore per il record,
- circa 43 byte per il record di INDI1,

• circa 35 byte per il record di INDI2.

Supponiamo di voler creare un archivio di 500 record; sono necessari:

• 500 settori per i record principali,

• 500*43=21500 byte per INDI1 in memoria, che diventano circa 80 settori sul dischetto,

• 500*35=17500 byte per INDI2 in memoria, ma questi non contano perchè si sovrappongono a quelli usati per INDI1; essi, però, contano sul dischetto per circa 70 settori.

In sostanza sul dischetto si occupano 1+500+80+70=651 settori, mentre in memoria occorrono circa 20000 byte per gli indici più le altre variabili. Sembra possibile arrivare a 500 record, ma devi tener presente che diventano molto lenti gli ordinamenti degli indici.Se vuoi gestire in modo sequenziale un file di questo tipo, devi leggerlo seguendo, record dopo record, il file indice INDI1.

3.3 ESEMPIO DI ARCHIVIO RANDOM/USER

In questo paragrafo mostriamo come si può ovviare all'inconveniente che abbiamo puntualizzato per l'esempio precedente, cioè il fatto della non possibilità di usare il comando VALIDATE per un dischetto che contenga registrazioni RANDOM.

Procediamo in questo modo:

• prepariamo un programma, di nome INIZIO, che serve solo per inizializzare l'archivio,

• esso crea un file sequenziale di tipo USER (USR), per distinguerlo dai normali sequenziali di tipo SEQ, registrando record logici esattamente di 254 caratteri, compresi i caratteri di fine campo (RETURN),

• il primo record logico contiene i dati generali del disco, completati con campi pieni di spazi per raggiungere 254 caratteri,

• dopo, ogni record logico è formato dagli stessi campi del record logico del file RANDOM, in modo da coprire completamente il settore, solo che i campi sono tutti formati dal carattere CHR$(99) seguito, eventualmente, a seconda della lunghezza, da spazi (codice ASCII 32),

• il sistema concatena i record usando i primi due caratteri di ogni settore,

• il file di tipo USR viene registrato nella directory con il nome che gli è stato assegnato, che è anche il nome assegnato al disco dati in fase di formattazione,

• il programma chiede a priori quanti record si vogliono, tale numero va deciso in fase di creazione del file, al limite restano poi dei record vuoti,

• dopo la chiusura del file di tipo USR, il programma va a leggere nella directory l'indirizzo assegnato al primo record del file,

• il programma comincia a scrivere su disco il primo record del file indice INDI1, lasciando in bianco la chiave e registrando l'indirizzo di traccia e settore, letto nella directory, questo primo record reca l'indirizzo del settore dove sono registrati i dati generali del disco,

• il programma legge in modo diretto (RANDOM) il primo blocco e scrive su disco il secondo record del file INDI1 (che si riferisce al primo record del file random), lasciando in bianco la chiave, ma registrando l'indirizzo di traccia e settore, e, procede così per il numero dei record che sono stati registrati nel file USR,

• alla fine sul discetto è registrato un file di tipo USR, che dopo verrà usato come RANDOM, ma senza dovere più nè allocare nè liberare blocchi,

• inoltre sul dischetto è registrato INDI1, con le chiavi in bianco e gli indirizzi registrati,

• per riconoscere i bloccchi liberi serve la chiave di INDI1, contenente solo il carattere CHR$(99), seguito da spazi,

• il disco dati può essere usato da un programma ottenuto da ARCHIRANDOM, modificandolo opportunamente,

• il dischetto dei dati può subire la funzione VALIDATE senza danni,

• qualora la struttura del record del file random non arrivi a coprire 254 caratteri, si aggiunge in fondo un campo di riempimento, infatti per ottenere che il file sequenziale usi un settore per ogni record logico, questo deve essere esattamente di 254 caratteri,

• quando il sistema crea un file sequenziale (anche USR), esso salta automaticamente la traccia 18.

Per migliorare ulteriormente il nostro programma di archivio ci mettiamo nelle condizioni di non doverlo modificare, come suggerito nel paragrafo precedente, se cambiamo la struttura del record. Per semplicità continuiamo a ragionare in termini di record logico che occupa un intero settore. Possiamo procedere così:

• prepariamo un programma di strutturazione dell'archivio, di nome PRESTRUTT,

• esso ci chiede:

1) il numero dei campi del record
2) la lunghezza e il nome di ogni campo,

• controlla che la somma delle lunghezze dei campi, compresi i caratteri di fine campo, non superi 254,

• scrive un file sequenziale con tutte le informazioni sulla struttura dell'archivio,

• tale file sequenziale, che chiamiamo STRUTTURA, viene letto, sia dal programma di inizializzazione, INIZIO, che dal programma di gestione dell'archivio, ARANDOMUSER, e fornisce i dati necessari per lavorare.

Segue il listato del programma PRESTRUTT.

```
5 REM PRESTRUTT
10 S1$="LA SOMMA DELLE LUNGHEZZE SUPERA 254"
15 S2$="MODIFICA QUALCHE CAMPO"
20 PRINT"□DEFINIZIONE STRUTTURA RECORD□"
25 REM CHIEDE NUMERO CAMPI MINORE DI 20
30 INPUT"NOME FILE: ";N$
35 INPUT"QUANTI CAMPI: ";NC
40 IFNC<2ORNC>20THEN20
45 PRINT"□NOMI E LUNGHEZZE CAMPI"
50 PRINT"OGNI CAMPO MINORE DI 80 CARATT."
55 PRINT"OGNI NOME MASSIMO 10 CARATTERI"
60 DIMD$(NC),L(NC)
65 FORK=1TONC
70 PRINTK;"NOME: ";:INPUTD$(K)
75 INPUT"LUNGHEZZA: ";L(K)
80 NEXTK
85 PRINT"□CONTROLLO CAMPI"
90 FORK=1TONC
95 IFLEN(D$(K))>10THEND$(K)=LEFT$(D$(K),10)
100 IFL(K)>79THENL(K)=79
105 PRINTK;D$(K);" ";L(K)
110 NEXTK
115 PRINT"□CONFERMI S/N ";:INPUTR$
120 IFR$="S"THEN145
125 INPUT"QUALE CAMPO: ";X
130 IFX<0ORX>NCTHEN125
135 INPUT"CAMPO: ";D$(X)
140 INPUT"LUNGHEZZA: ";L(X):GOTO85
145 REM CALCOLA LUNGHEZZA RECORD
150 S=NC:FORK=1TONC
155 S=S+L(K):NEXTK
160 IFS>254THENPRINTS1$:PRINTS2$:GOTO90
165 PRINT"NOME FILE: ";N$
170 PRINT"LUNGHEZZA RECORD: ";S
175 OPEN15,8,15,"I"
180 OPEN2,8,2,"STRUTTURA,S,W"
185 PRINT#2,N$
190 PRINT#2,NC
195 FORK=1TONC:PRINT#2,D$(K):NEXTK
200 FORK=1TONC:PRINT#2,L(K):NEXTK
205 CLOSE2
210 PRINT"□VERIFICA"
```

```
215 OPEN2,8,2,"STRUTTURA,S,R"
220 INPUT#2,A$
225 PRINT"NOME FILE: ";A$
230 INPUT#2,X
235 PRINT"NUMERO CAMPI: "X
240 FORK=1TOX:INPUT#2,D$(K):NEXTK
245 FORK=1TOX:INPUT#2,L(K):NEXTK
250 CLOSE2
255 FORK=1TOX:PRINTK;D$(K);L(K):NEXTK
260 STOP
```

Il programma chiede:

- il nome da assegnare al file archivio, linea 30,
- il numero dei campi e controlla che tale numero non superi 20 e non sia inferiore a 2 (per fare entrare la descrizione nel video, un campo per linea), linee 35-40,
- il nome e la lunghezza di ogni campo, con possibilità di correzione, linee 45-140,
- se la somma delle lunghezze, compresi i caratteri di fine campo, supera 254, chiede di modificare qualche campo, linee 145-160
- taglia le descrizioni dei campi a 10 caratteri, se più lunghi, linea 95,
- taglia la lunghezza dei campi a 79, se maggiore, linea 100,
- se tutto va bene, scrivi sul dischetto dei programmi (che quindi non deve essere protetto da scrittura), il file di nome STRUTTURA, di tipo SEQ, linee 175-205,
- il file STRUTTURA contiene:

  - nome del file,
  - numero dei campi del record,
  - nome di ogni campo,
  - lunghezza di ogni campo.

Alla fine il programma ripropone, per controllo, sul video il contenuto del file STRUTTURA generato, linee 210-260. Il file STRUTTURA viene memorizzato sul disco del programma.

Vediamo ora come è stato costruito il programma INIZIO:

- legge inizialmente i dati del file STRUTTURA, mostra sul video il nome del file e il numero dei campi, poi chiede il numero N dei record da registrare per il file, ed elenca i nomi e le lunghezze dei campi, linee 10-105,
- controlla che la somma delle lunghezze non superi 254, se supera si ha uno STOP, linee 110-120,

• prepara i campi dati, CHR$(99)+spazi, linee 125-145,

• se la somma delle lunghezze è minore di 254, prepara un campo di lunghezza opportuna per riempire il settore, linee 150-170,

• chiede di montare un disco dati nuovo da formattare e preparare per l'archivio, linee 175-190,

• formatta il dischetto dati e apre un file sequenziale con il nome fornito dal file STRUTTURA e il tipo USR, linee 191-210,

• comincia a scrivere sul file il primo record relativo ai dati generali del disco, linee 215-240, che sono:

- nome file,
- data,
- numero N di record previsto (massimo),
- numero record realmente esistente (inizialmente 0),

• scrive il numero N di record richiesto, seguendo il tracciato rilevato dal file STRUTTURA, ma scrivendo campi formati da CHR$(99) seguito da spazi (o solo da CHR$(99)), linee 245-270,

• chiude il file, che rimane registrato nella directory, con tutti i settori concatenati usando i primi due byte, linea 275,

• il file occupa N+1 blocchi,

• apre un canale per lettura diretta, il canale 2, linee 280-295,

• legge in modo diretto (random) il settore 18,1 (primo della dyrectory) e, con il puntatore a 3, va a leggere l'indirizzo del primo settore usato, tale indirizzo è quello del blocco usato per i dati generali del disco, e chiude il canale 2, linee 300-35,

• apre il file INDI1 sul canale 3, e un canale ad accesso diretto il 2, linee 340-380,

• scrive il primo record di INDI1, quello relativo ai dati generali del disco, linee 385-390,

• legge in modo diretto gli N settori concatenati, prelevando dalle prime due posizioni di ogni blocco l'indirizzo di traccia e settore, e scrive il corrispondente record di INDI1, linee 395-430,

• i campi chiave di INDI1 contengono la somma dei primi due campi del record (iniziano con CHR$(99)),

• chiude i file, linee 435-440,

• esegue la VALIDATE del disco, linea 443.

```
5 REM INIZIO
10 REM DIMENSIONA COSTANTI E VARIABILI
15 REM LEGGE FILE STRUTTURA
20 CH$=CHR$(13)
25 S1$=" ":FORK=1TO39:S1$=S1$+" ":NEXTK
```

```
30 S2$="              "
35 OPEN15,8,15,"I"
40 OPEN2,8,2,"STRUTTURA,S,R":GOSUB445
45 INPUT#2,N$,NC:GOSUB445
50 N$=LEFT$(N$+S1$,16)
55 PRINT"DATI FILE STRUTTURA"
60 PRINT"FILE: ";N$;" NUMERO CAMPI: ";NC
65 INPUT"QUANTI RECORD: ";N
70 IFN>500THEN STOP
75 NX$=STR$(N)
80 NX$=RIGHT$(S2$+NX$,6)
85 DIMD$(NC),L(NC),Y$(NC)
90 FORK=1TONC:INPUT#2,D$(K):NEXTK
95 FORK=1TONC:INPUT#2,L(K):NEXTK
100 CLOSE2
105 FORK=1TONC:PRINTD$(K);"    ";L(K):NEXTK
110 SR=NC:FORK=1TONC:SR=SR+L(K):NEXTK
115 PRINT"LUNGHEZZA RECORD: ";SR
120 IFSR>254THENPRINT"NON POSSO PROSEGUIRE":STOP
125 REM CAMPI DATI CON CHR$(99) + SPAZI
130 FORK=1TONC:Y$(K)=CHR$(99):IFL(K)=1THEN145
135 FORJ=1TOL(K)-1
140 Y$(K)=Y$(K)+CHR$(32):NEXTJ
145 NEXTK
150 IFSR=254THEN175
155 REM CREA CAMPO FINALE SE SR<254
160 IFSR=253THENDN$="":GOTO175
165 DN$=CHR$(99):FORK=1TO253-SR-1
170 DN$=DN$+CHR$(32):NEXTK
175 REM CREA FILE USER
180 PRINT"MONTA DISCO DATI"
185 PRINT"PREMI UN TASTO PER PROSEGUIRE"
190 R$="":GETR$:IFR$=""THEN190
191 PRINT"PAZIENZA, ATTENDI, STO PREPARANDO"
192 PRINT"IL DISCO DATI"
195 CLOSE15:OPEN15,8,15,"N0:"+N$+",99"
200 REM RIEMPIE RECORD
205 REM CON CAMPI CHE INIZIANO CON CHR$(99)
210 OPEN2,8,2,N$+",U,W":GOSUB445
215 REM SCRIVE PRIMO RECORD PER DATI DISCO
220 REM OCCUPANDO UN INTERO SETTORE
225 PRINT#2,N$CH$"GGMMAA"CH$NX$CH$"0"CH$" "CH$" "CH$;
```

```
230 GOSUB445
235 PRINT#2,S1$CH$S1$CH$S1$CH$S1$CH$S1$CH$S2$
240 GOSUB445
245 FORK=1TON
250 FORJ=1TONC
255 PRINT#2,Y$(J);CH$;
260 GOSUB445
265 NEXTJ
266 IFSR<254THENPRINT#2,DN$
270 NEXTK
275 CLOSE2
280 REM CREA FILE INDICE
285 REM INDIRIZZO PRIMO SETTORE DIRECTORY
290 CLOSE15:OPEN15,8,15,"I"
295 OPEN2,8,2,"#":GOSUB445
300 REM LEGGE SETTORE 18,1
305 PRINT#15,"U1:2,0,18,1":GOSUB445
310 REM PUNTATORE A TRACCIA E SETTORE
315 PRINT#15,"B-P:2,3":GOSUB445
320 T$="":S$="":GET#2,T$,S$
325 T$=T$+CHR$(0):S$=S$+CHR$(0)
330 CLOSE2
335 T=ASC(T$):S=ASC(S$)
340 PRINT"PRIMO BLOCCO"
345 PRINT"TRACCIA: ";T;"  SETTORE: ";S
350 PRINT"PREMI UN TASTO PER PROSEGUIRE"
355 R$="":GETR$:IFR$=""THEN355
360 REM GENERA INDICE
365 REM LEGGENDO FILE USR COME RANDOM
370 CLOSE15:OPEN15,8,15,"I"
375 OPEN2,8,2,"#":GOSUB445
380 OPEN3,8,3,"INDI1,S,W":GOSUB445
385 REM SCRIVE PRIMO RECORD INDI1
390 PRINT#3,Y$(1)+Y$(2);CH$;T;CH$;S:GOSUB445
395 REM SCRIVE ALTRI N RECORD
400 FORK=1TON
405 PRINT#15,"U1:2,0";T;S:GOSUB445
410 PRINT#15,"B-P:2,0":GOSUB445
415 GET#2,T$,S$:GOSUB445
416 T=ASC(T$+CHR$(0)):S=ASC(S$+CHR$(0))
420 PRINTT,S
```

```
425 PRINT#3,Y$(1)+Y$(2);CH$;T;CH$;S:GOSUB445
430 NEXTK
435 PRINT"IL DISCO DATI E' PRONTO"
440 CLOSE2:CLOSE3:CLOSE15
443 OPEN15,8,15,"V":CLOSE15:STOP
445 INPUT#15,EN,EM$,ET,ES
450 IFEN=0THENRETURN
455 PRINT"ERRORE DISCO":PRINTEN,EM$,ET,ES
460 STOP
```

Il dischetto dati, prodotto da INIZIO, in collaborazione con PRESTRUTT, è sicuro, cioè tutto è registrato nella directory.

Puoi provare a listare INDI1 con il programma CONTRINDI1, che segue; così puoi vedere come il sistema usa i settori del dischetto, partendo da 17,0 e arrivando alla traccia 1, per proseguire poi dalla traccia 19.

```
10 REM CONTRINDI1
20 OPEN15,8,15,"I"
30 OPEN2,8,2,"INDI1,S,R"
35 OPEN4,4:PRINT#4,"CONTROLLO INDI1":PRINT#4
40 K=0:I=0
45 INPUT#2,A$,T,S:TS=ST:K=K+1
50 PRINT#4,K;T;S;" ";
60 I=I+1:IFI=5THENPRINT#4:I=0
70 IFTS=64THENPRINT#4:CLOSE2:CLOSE4:CLOSE15:STOP
80 GOTO45
```

Noi abbiamo provato a generare un file dati, di nome PROVE, con 500 record, la directory del dischetto è riportata qui:

```
0 "PROVE                    " 99 2A
501   "PROVE                " USR
90    "INDI1"               SEQ
73 BLOCKS FREE.
```

come puoi vedere, con 500 record restano pochi blocchi per l'indice secondario; i nostri conti precedenti erano stati troppo ottimisti, può funzionare se si crea un indice secondario per un campo chiave di pochi caratteri.Si potrebbe risparmiare spazio su disco comprimendo l'indice, cioè riducendo il record logico da 3 campi a un campo, senza caratteri separatori, e memorizzando il numero di traccia e il numero di settore come una sola coppia di byte (CHR$) in coda alla chiave. Naturalmente gli altri programmi che usano INDI1 dovrebbero tenere conto di questa struttura. Noi abbiamo memorizzato traccia e settore come numeri, occupando più spazio del necessario.

Puoi, servendoti del programma di utilità DISPLAY T&S, stampare il contenuto di qualche blocco, per controllare come ha lavorato il programma INIZIO.

Ora dobbiamo vedere come trasformare il programma ARCHIRANDOM, esposto nel paragrafo precedente, per farlo lavorare su un dischetto dati preparato con i programmi PRESTRUTT e INIZIO. Il nuovo programma per trattare i file RANDOM/USER si chiama ARANDOMUSER.

Le principali differenze rispetto a ARCHIRANDOM sono le seguenti:

- il settore dei dati generali del disco non è più quello di indirizzo 1,0, ma il suo indirizzo si trova leggendo il primo record del file INDI1,
- il file INDI1 deve sempre essere caricato nei tre vettori dell'indice, qualunque scelta si faccia, a partire dal secondo record (il file ha N+1 record),
- la fase di inizializzazione ex-novo dell'arcrchivio sparisce dal programma,
- tra i dati generali del disco ne figurano uno in più e due in meno, infatti si ha il numero N dei record previsti al massimo, e il numero K dei record effettivamente presenti, mentre non servono più la traccia e il settore dell'ultimo record scritto,
- il file indice INDI1 deve sempre essere ordinato e riscritto tutto, altrimenti si perdono gli indirizzi dei record,
- nell'ordinamento del file indice in memoria, i record non usati, che iniziano con CHR$(99), vanno in fondo, ma restano,
- per cancellare un record, si pone CHR$(99) seguito da spazi nella chiave di INDI1, per uniformarsi agli altri record vuoti,
- il file INDI2 si riferisce solo ai record effettivamente presenti K, ma per operare correttamente si deve lavorare con un indice principale ordinato,
- quando si aggiunge un nuovo record, si deve cercare nell'indice la prima chiave che inizia con CHR$(99), leggere il corrispondente settore e riscriverlo con i nuovi dati; non si devono infatti perdere i primi due caratteri di concatenamento.

Segue il listato di ARANDOMUSER.

```
5 REM ARANDOMUSER
10 REM-----------------------------------
15 REM DEFINIZIONE COSTANTI E VARIABILI
20 REM-----------------------------------
25 S1$="DATA ULTIMO AGG.       "
30 S2$="FINITO SPAZIO ASSEGNATO"
35 SP$="                                "
40 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(32)+CHR$(32)
45 REM **LEGGE DATI DA FILE STRUTTURA**
50 REM ******************************
55 OPEN15,8,15,"I"
57 PRINT"□□□"TAB(8)"MONTA DISCO FILE STRUTTURA"
58 GOSUB610
60 OPEN2,8,2,"STRUTTURA,S,R":GOSUB525
65 INPUT#2,N$,NC:GOSUB525
70 DIMD$(NC):DIMY$(NC):DIML(NC)
75 FORK=1TONC:INPUT#2,D$(K):NEXTK
80 FORK=1TONC:INPUT#2,L(K):NEXTK
85 CLOSE2
90 PRINT"□□□"TAB(10)"MONTA DISCO DATI"
95 GOSUB610
100 CLOSE15:OPEN15,8,15,"I"
105 REM **LEGGE DATI DISCO**
110 REM ******************
115 OPEN10,8,10,"INDI1,S,R":GOSUB525
120 REM **SETTORE DATI GENERALI**
125 REM ************************
130 INPUT#10,A$,TD,SD
135 GOSUB455
140 REM **DIMENSIONAMENTI PER INDICE**
145 REM ****************************
150 DIMC$(N),T%(N),S%(N)
155 REM **LETTURA INDICE**
160 REM *****************
165 GOSUB510:GOSUB555
170 REM **PRESENTAZIONE MENU' E SCELTE**
175 REM ******************************
180 PRINT"□□□";TAB(10);"GESTIONE ARCHIVIO":PRINT
185 PRINTTAB(10)"1=AGGIORNAMENTO"
190 PRINTTAB(10)"2=LISTA PRINCIPALE"
195 PRINTTAB(10)"3=CREAZ.IND.SEC."
200 PRINTTAB(10)"4=LISTA SECONDARIA"
```

```
205 PRINTTAB(10)"9=FINE"
210 INPUT"COSA SCEGLI ";X
215 IFX<1ORX>4ANDX<>9THENPRINT"□";:GOTO210
220 PRINT:PRINT"PRESENTI ";K;" RECORD"
225 PRINTS1$;G1$;"/";M1$;"/";A1$
226 GOSUB610
230 IFX=9THEN1230
235 ONXGOTO705,1035,1120,1180
240 STOP
245 REM **INGRESSO DATI**
250 REM ****************
255 PRINT"□INGRESSO DATI"
260 FORJ=1TONC:Y$(J)="":NEXTJ
265 PRINT"PER USCIRE $ PER COGNOME";
270 PRINT"■SE MANCANO DATIPREMI SOLO RETURN"
275 PRINTD$(1);:INPUTY$(1)
280 IFY$(1)="$"THENW=1:RETURN
285 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
290 REM **SISTEMA LUNGHEZZA DATI**
295 REM ***************************
300 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
305 NEXTJ:GOSUB900:RETURN
310 REM **SCRITTURA INDICE PRINCIPALE**
315 REM ********************************
320 PRINT#15,"S0:INDI1":PRINT#15,"I"
325 OPEN10,8,10,"INDI1,S,W":GOSUB525
330 PRINT#10,LIM$;CH$;TD;CH$;SD;CH$;:GOSUB525
335 FORJ=1TON
340 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
345 GOSUB525:NEXTJ
350 CLOSE10:RETURN
355 REM **SCRITTURA NEL BUFFER**
360 REM ************************
365 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
370 PRINT#11,Y$(J);CH$;:GOSUB525
375 NEXTJ:RETURN
380 REM **LEGGI SETTORE NEL BUFFER**
385 REM ****************************
390 PRINT#15,"U1:11,0";T;S:GOSUB525:RETURN
395 REM **PUNTATORE NEL BUFFER**
400 REM ************************
405 PRINT#15,"B-P:"11;2:GOSUB525:RETURN
```

```
410 REM **SCRITTURA RECORD**
415 REM ********************
420 PRINT#15,"U2:"11;0;T;S:GOSUB525:RETURN
425 REM **LETTURA RECORD**
430 REM ******************
435 PRINT#15,"I":OPEN11,8,11,"#":GOSUB525
440 PRINT#15,"U1:"11;0;T;S:GOSUB525:GOSUB405
445 FORJ=1TONC:INPUT#11,Y$(J):GOSUB525
450 NEXTJ:CLOSE11:RETURN
455 REM **LETT.DATI DISCO**
460 REM ********************
465 OPEN11,8,11,"#":GOSUB525
470 PRINT#15,"U1:"11;0;TD;SD:GOSUB525
475 GOSUB395:INPUT#11,N$,R$,N,K:GOSUB525
480 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
485 A1$=RIGHT$(R$,2):CLOSE11:RETURN
490 REM **LETT. INDICE**
495 REM ****************
496 CLOSE10:OPEN10,8,10,"INDI1,S,R":GOSUB525
497 INPUT#10,A$,TD,SD
500 REM **DA SECONDO RECORD**
505 REM *********************
510 FORJ=1TON
515 INPUT#10,C$(J),T%(J),S%(J):GOSUB525
520 NEXTJ:CLOSE10:RETURN
525 REM **ROUTINE ERRORE**
530 REM ******************
535 INPUT#15,EN,EM$,ET,ES
540 IFEN=0THENRETURN
545 PRINT"ERRORE DISCO"
550 PRINTEN,EM$,ET,ES:CLOSE15:STOP
555 REM **DATA DISCO**
560 REM **************
565 PRINT"DATA PER DISCO"
570 INPUT"  GG,MM,AA■■■■■■■■■■■■";G$,M$,A$:RETURN
575 REM **SCRITT. IND. SEC.**
580 REM *********************
585 PRINT#15,"S:INDI2":PRINT#15,"I"
590 OPEN10,8,10,"INDI2,S,W"
595 FORJ=1TOK
600 PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;
605 GOSUB525:NEXTJ:CLOSE10:RETURN
```

```
608 REM **ATTESA TASTO**
609 REM ****************
610 R$="":GETR$:IFR$=""THEN610
615 RETURN
620 REM **RICERCA POSTO LIBERO**
625 REM ************************
630 FORI=1TON
635 IFLEFT$(C$(I),1)<>CHR$(99)THEN645
640 T=T%(I):S=S%(I):JJ=I:I=N
645 NEXTI:RETURN
650 REM **AGGIUNTA NUOVI RECORD**
655 REM *************************
660 GOSUB245:GOSUB620
665 IFW=1THENK=K-1:CLOSE11:GOTO1230
670 GOSUB380:GOSUB395:GOSUB355
675 REM **AGGIORNAMENTO INDICE**
680 REM ************************
685 C$(JJ)=Y$(1)+Y$(2)
690 GOSUB410:K=K+1
695 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO1230
700 GOTO660
705 REM---------------
710 REM AGGIORNAMENTO
715 REM---------------
720 PRINTTAB(10);"AGGIORNAMENTO ARCHIVIO"
725 PRINT:PRINTTAB(10);"1=CORREZIONE"
730 PRINTTAB(10);"2=AGGIUNTA ELEM."
735 PRINTTAB(10);"3=CANCELL.ELEM."
740 PRINTTAB(10);"9=FINE"
745 INPUT"COSA SCEGLI ";X:IFX=9THEN1230
750 IFX<1ORX>3THEN720
755 IFX=2THEN950
760 IFX=3THEN990
765 GOTO850
770 REM **RICERCA RECORD**
775 REM ******************
780 PRINT"";D$(1);:INPUTY$(1):W=0
785 IFY$(1)="$"THENW=1:RETURN
790 PRINTD$(2);:INPUTY$(2)
795 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
800 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
```

```
805 INPUT"QUALE OCCORRENZA ";X
810 IFX<=0THENPRINT"□";:GOTO805
815 FORJ=1TON:IFC$(J)=Y$(1)+Y$(2)THEN835
820 NEXTJ
825 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
830 GOSUB610:GOTO770
835 IFX<>1THENX=X-1:GOTO820
840 T=T%(J):S=S%(J)
845 I=J:J=N:NEXTJ:RETURN
850 REM **CORREZIONE**
855 REM ***************
860 GOSUB770:IFW=1THEN720
865 GOSUB425
870 REM **VA A MODIFICA DATI**
875 REM **********************
880 GOSUB900
885 PRINT#15,"I":OPEN11,8,11,"#":GOSUB395:GOSUB355
890 C$(I)=Y$(1)+Y$(2):T%(I)=T:S%(I)=S
895 GOSUB410:CLOSE11:GOTO850
900 REM **CONTROLLO DATI E MODIFICHE**
905 REM ******************************
910 PRINT"□CONTROLLO DATI E MODIFICHE"
915 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
920 INPUT"□TUTTO BENE S/N ";X$:IFX$="S"THENRETURN
925 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
930 IFX=0THENRETURN
935 IFX>NCORX<1THENPRINT"□":GOTO920
940 Y$(X)=""
943 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
945 GOTO910
950 REM **AGGIUNTA**
955 REM ************
960 OPEN11,8,11,"#":GOSUB525
965 PRINT"□AGGIUNTA NUOVI RECORD"
970 PRINT"PRESENTI ";K;" RECORD"
975 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
980 GOSUB610
985 W=0:K=K+1:GOTO660
990 REM **CANCELLAZIONE**
```

```
995 REM ****************
1000 M=0
1005 GOSUB770:IFW=1THEN1025
1010 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
1015 M=M+1
1020 GOTO1005
1025 REM SIST. INDICE
1030 GOSUB1295:K=K-M:GOTO1255
1035 REM------------
1040 REM LISTA FILE
1045 REM------------
1050 T$="LISTA PER INDICE PRIM."
1055 PRINT"ACCENDI STAMPANTE     PREMI UN TASTO"
1060 GOSUB610
1065 PRINT"□";T$:OPEN4,4
1070 PRINT#4:PRINT#4:PRINT#4,T$
1075 PRINT#4:PRINT#4
1080 FORI=1TOK:T=T%(I):S=S%(I):GOSUB425
1085 PRINT#4,Y$(1);"  ";Y$(2)
1090 IFNC=2THEN1100
1095 FORM=3TONC:PRINT#4,D$(M)": "Y$(M):NEXTM
1100 PRINT#4:PRINT#4
1110 NEXTI
1115 CLOSE4:GOTO1280
1120 REM-------------------------------
1125 REM CREAZIONE INDICE SECONDARIO
1130 REM-------------------------------
1135 PRINT"□INDICE SECONDARIO"
1140 INPUT"□QUALE CAMPO ";X$
1145 X=VAL(X$):IFX<2ORX>NCTHENGOTO1140
1155 FORI=1TOK:T=T%(I):S=S%(I):GOSUB425
1156 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
1160 C$(I)=Y$(X):NEXTI:GOSUB1365
1165 GOSUB575
1170 PRINT"FINITO IND. SEC."
1175 GOTO1280
1180 REM-------------------------------
1185 REM LISTA PER INDICE SECONDARIO
1190 REM-------------------------------
1195 PRINT"□LISTA IND. SEC."
1200 OPEN10,8,10,"INDI2,S,R":GOSUB525
```

```
1205 FORJ=1TOK:INPUT#10,C$(J),T%(J),S%(J)
1210 GOSUB525:NEXTJ
1215 CLOSE10
1220 T$="LISTA IND. SEC. "
1225 GOTO1055
1230 REM **CHIUSURA**
1235 REM ************
1240 PRINT"CHIUSURA ARCHIVIO".
1245 PRINT"IND. PRINC. DA ORDIN. S/N ":INPUTR$
1250 IFR$="S"THENGOSUB1295
1255 GOSUB310:OPEN11,8,11,"#":GOSUB525
1260 PRINT#15,"U1:"11;0;TD;SD:GOSUB525:GOSUB395
1265 PRINT#11,N$CH$G$M$A$CH$;N;CH$;K;CH$;
1270 GOSUB525
1275 PRINT#15,"U2:"11;0;TD;SD:GOSUB525:CLOSE11
1280 PRINT"FINITO AGGIORNAMENTO"
1285 PRINT"SONO PRESENTI ";K;" RECORD"
1290 CLOSE15:STOP
1295 REM **ORDINAMENTO CRESCENTE**
1300 REM ************************
1305 L=N-1
1310 W=0
1315 FORJ=1TOL
1320 IFC$(J)<=C$(J+1)THEN1345
1325 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
1330 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
1335 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
1340 W=1
1345 NEXTJ
1350 IFW=0THENRETURN
1355 IFL=1THENRETURN
1360 L=L-1:GOTO1310
1365 REM **ORDINAMENTO DECRESCENTE**
1370 REM ***************************
1375 L=K-1
1380 W=0
1385 FORJ=1TOL
1390 IFC$(J)>=C$(J+1)THEN1415
1395 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
1400 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
1405 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
```

```
1410 W=1
1415 NEXTJ
1420 IFW=0THENRETURN
1425 IFL=1THENRETURN
1430 L=L-1:GOTO1380
```

Non ripetiamo la lista delle variabili usate, infatti il programma è stato costruito modificando il precedente, e le differenze sono poche. Segnaliamo le variabili TD e SD, che contengono l'indirizzo del settore utilizzato per i dati generali del disco, e che deve essere aggiornato prima di chiudere. Ricorda che SP$ (LINEA 35) deve contenere un numero di spazi pari al campo più lungo del RECORD. Noi abbiamo usato 30 spazi, se ne usi 79 va sempre bene.

Anche questo programma potrebbe essere migliorato, infatti ogni funzione termina con la chiusura, e dato che il file guida, STRUTTURA, sta sul disco dei programmi, è necessario cambiare disco dopo aver dato nuovamente il RUN. Puoi modificare la linea 1290 così:

1290 CLOSE15:GOTO90

in modo che finita una fase ricompare la domanda MONTA DISCO DATI, come misura di prudenza per verificare che è montato il disco giusto. In questo modo non vengono persi i dati letti dal file STRUTTURA, come succede scrivendo RUN, e non si deve rimontare il disco dei programmi.

Altro perfezionamento, utile se il file è formato da molti record, può essere quello di creare uno SWITCH, che viene messo a zero, quando viene letto in memoria INDI1, e viene posto a 1 se si operano aggiunte, cancellazioni o modifiche dei campi chiave dei record. Alla fine se lo switch è rimasto a zero, non è necessario nè riordinare l'indice, nè riscriverlo sul dischetto, infatti non è variato.

Con file formati da molti record risulta necessario sostituire le routine di ordinamento con altre più veloci.
Un'ultima considerazione riguardo all'indice del file, qualora i record logici fossero più corti, per esempio 127 caratteri, in modo da poterne scrivere due in un settore, nell'indice, oltre all'indirizzo del settore, dovrebbe comparire l'indicazione 1 o 2, o altro, per individuare il record. Inoltre per modificare o scrivere un record si dovrebbe sempre leggere tutto il settore , andare a modificare la parte che interessa, e poi riscrivere tutto il settore.Un file di tipo RANDOM/USER, come quello costruito da noi, può anche essere gestito come file sequenziale, senza ricorrere all'indice INDI1. La sequenza dei record dipende dal caricamento iniziale, dalle cancellazioni e dalle aggiunte.

CAPITOLO 4

FILE RELATIVI

4.1 FILE RELATIVI DI DATI

I FILE RELATIVI sono organizzati in modo tale, che, per accedere a un record logico, si deve fornire al DOS il suo numero d'ordine nel file. I record logici devono essere di lunghezza fissa. Il sistema, conoscendo la lunghezza in caratteri di un record logico e il suo numero d'ordine nel file, può, con un semplice algoritmo, determinare in quale blocco fisico inizia il record e quindi accedere ad esso, sia in lettura che in scrittura. Per poter lavorare così è necessario che i record logici siano tutti della stessa lunghezza, ma non è necessario che la loro dimensione sia un sottomultiplo di un settore. Sono consentiti record logici di lunghezza a piacere, purchè minore o uguale a 254, compresi i caratteri separatori di campo e di record. Un record logico può, come del resto nei file sequenziali, stare a cavallo di due settori (spanned).

Il sistema, per gestire i file relativi, usa la tecnica che descriviamo:

- Quando crea il file costruisce un suo indice interno, chiamato SIDE SECTOR. Esso consiste in un massimo di 6 settori, numerati logicamente da 0 a 5, che servono per contenere l'indice di tutti i blocchi fisici (settori) utilizzati per il file relativo. In ogni blocco SIDE SECTOR possono essere registrati gli indirizzi di 120 settori; per questa ragione un file relativo può occupare al massimo 120*6=720 settori. Per i nostri dischetti tale numero è sovrabbondante, dato che essi dispongono solo di 664 settori, e da questi dobbiamo toglierne 6 per i SIDE SECTOR. Il numero dei record logici gestibili non può superare 65535, infatti il numero di record logico si esprime in due byte. La struttura di un SIDE SECTOR è la seguente:

byte 0 e 1, concatenamento al settore successivo
byte 2, numero del side sector, da 0 a 5
byte 3, lunghezza del record logico, massimo 254
byte 4 e 5, traccia e settore del side sector 0
byte 6 e 7, traccia e settore del side sector 1
byte 8 e 9, traccia e settore del side sector 2
byte 10 e 11, traccia e settore del side sector 3
byte 12 e 13, traccia e settore del side sector 4
byte 14 e 15, traccia e settore del side sector 5
byte 16 e 17, indirizzo primo settore dati
byte 18 e 19, indirizzo secondo settore dati
...
...
byte 254 e 255, indirizzo centoventesimo settore dati.

• I settori usati per i dati sono, come sempre, concatenati usando i primi due byte, e ogni settore può contenere 254 caratteri di dati.

• Viene registrata una entrata nella directory, che indica il numero totale dei blocchi occupati, comprendendo i blocchi per i dati e quelli occupati per i side sector. In questa entrata sono usati 3 byte per specificare l'indirizzo del primo side sector e la lunghezza del record logico.

• Quando si crea il file, i record logici non usati sono riempiti dal sistema ponendo FFH (255 in esadecimale) nella prima posizione e zeri binari (00H) nelle successive. I record logici sono così marcati a priori.

Per gestire un file relativo il sistema usa un buffer in più, tre invece di due, dato che deve gestire anche i side sector. Diminuisce così il numero dei file che si possono trattare contemporaneamente.

Vediamo i comandi necessari per gestire i file relativi.

APERTURA DEL CANALE COMANDI:

OPEN15,8,15

come sempre.

APERTURA DEL FILE E DEL CANALE PER I DATI:

OPEN#lfn,8,sa,fn+",L,"+CHR$(LU)

dove:
lfn, numero logico del file, che noi, per abitudine, facciamo coincidere con il numero di canale sa,
8, numero della periferica,
sa, numero del canale, da 2 a 14,
fn, nome del file, che può essere preceduto da "0:" per l'unità,
L, parametro richiesto, precede la lunghezza del record logico,
LU, lungheza del record logico, passata come carattere con CHR$, minore di 255.

Nella OPEN non deve essere specificato il tipo di operazione, infatti si può sia leggere che scrivere, intercalando le operazioni.

LOCALIZZAZIONE DEL RECORD E, EVENTUALMENTE, DEL CAMPO NEL RECORD:

PRINT#15,stringa-comando

dove la stringa-comando si costruisce così:

"P"+CHR$(sa)+CHR$(LO)+CHR$(HI)+CHR$(BI)
dove:
P, sta per puntatore
sa, è il numero del canale usato nella OPEN
LO e HI, sono rispettivamente il byte basso e il byte alto del numero del record; si calcola:
HI=INT(numrec/256)
LO=numrec-HI*256
BI, è il puntatore al byte nel record, cioè al primo byte di un campo: 1 per il primo campo, N per un campo che si trovi dopo i primi N-1 caratteri dall'inizio del record. Questa PRINT#15, deve esssere usata per puntare a un record e a una posizione; dopo si può procedere con PRINT#lfn, con INPUT#lfn o con GET#lfn, andando in sequenza. Si deve ridefinire una posizione se si salta altrove nel file.

PER SCRIVERE DATI:

PRINT#lfn, lista-dati

con le solite regole, già viste, relative ai separatori tra le variabili e ai separatori da registrare. Devi però fare attenzione a quanto segue:

- dopo aver posizionato il puntatore, la lista-dati che usi con la PRINT#lfn, deve comprendere tutti i campi che vuoi scrivere, uno di seguito all'altro in quel particolare record; devi cioè usare una sola PRINT per scrivere in un record,

• se usi diverse PRINT# successive, il dato viene via via scritto in record diversi, successivi a quello puntato con la PRINT#15,

• naturalmente puoi usare diverse PRINT# per scrivere in uno stesso record, ma devi riposizionare il puntatore, al record e alla posizione, prima di scrivere ogni campo.

PER LEGGERE DATI:

INPUT#lfn, lista-dati

con le solite regole riguardo alla lunghezza dei campi, al loro tipo, e ai caratteri separatori.

Per leggere carattere per carattere:

GET#lfn, lista-dati

per leggere un carattere in ogni variabile, che è bene sia di tipo stringa.

PER CHIUDERE IL FILE:

CLOSElfn e poi **CLOSE15**.

La routine di errore, in fase di preestensione del file, deve ammettere il codice di errore 50.

Fare la PREESTENSIONE DEL FILE significa aprire il file e creare tutti i record in modo DUMMY, cioè riempiendoli con caratteri di comodo. Se si fa inizialmente questa operazione, la gestione del file risulta più veloce, infatti sono creati subito tutti i side sector necessari. Per fare la preestensione, basta aprire il file e andare a scrivere l'ultimo record; infatti il sistema per poter scrivere l'ultimo record deve predisporre tutti quelli che vengono prima. I record sono predisposti dal sistema ponendo FFH come primo carattere e continuando con 00H, senza separazione tra i campi, che esso non conosce. Qualora l'ultimo blocco dati (settore) non sia occupato tutto, il sistema lo completa con record DUMMY, pur registrando nel secondo byte la reale occupazione.

Abbiamo preparato dei programmi esempio. Iniziamo con CREAREL; esso crea un file di nome PROVA REL, formato da 30 record logici di 21 caratteri ciascuno. A questo punto non interessa la suddivisione in campi del record logico. Il programma crea il file e lo preestende andando a scrivere, a partire dal fondo 30 record formati da 20 lineette più il carattere CHR$(13).

```
5 REM CREAREL
10 REM CREAZIONE FILE RELATIVO
15 REM PREESTENSIONE FILE
20 REM 30 RECORD DI 21 CARATTERI
25 REM COMPRESO RETURN FINALE
30 DR$="0":NF$=DR$+":PROVA REL,L,"
35 SA=2:LF=2:RC$="----------":RC$=RC$+RC$
40 BI=1:NR=30:LU=21
45 OPEN15,8,15,"I"
50 REM APERTURA FILE
55 OPENLF,8,SA,NF$+CHR$(LU)
60 GOSUB135
65 REM PREESTENSIONE FILE
70 REM PER NR RECORD
75 FORX=NRTO1STEP-1
80 HI=INT(X/256):LO=X-HI*256
85 C$="P"+CHR$(SA)
90 REM NUMERO RECORD (BYTE BASSO + BYTE ALTO)
95 C$=C$+CHR$(LO)+CHR$(HI)
100 REM PUNTATORE AL PRIMO BYTE DEL RECORD
105 C$=C$+CHR$(BI)
110 REM POSIZIONA IL PUNTATORE
115 PRINT#15,C$:GOSUB135
120 REM SCRIVE IL RECORD DI LINEETTE
125 PRINT#LF,RC$:GOSUB135
130 NEXTX:CLOSELF:GOSUB135:CLOSE15:END
135 INPUT#15,EN,EM$,ET,ES
140 IFEN=0OREN=50THENRETURN
145 PRINTEN;EM$;ET;ES
150 STOP:RETURN
```

Il programma opera così:

- Prepara le costanti necessarie, lf sta per numero logico del file (non si può usare LFN, dato che FN è una parola chiave). Pone sa e lf a 2, LU=21 per lunghezza record logico, RC$ contiene 20 lineette, BI=1 per puntare al primo carattere del record, NR=30 per numero di record, linee 5-40.
- Apre il canale 15 e inizializza il dischetto, linea 45.
- Apre il file PROVA REL, linee 50-60.
- Preestende il file scrivendo record di lineette. Nota come vengono svolti i

calcoli, dato che si parte dal fondo, ogni volta deve essere ricalcolato il numero del record, e deve essere eseguita l'operazione di posizionamento del puntatore. Linee 65-130.

Noi, per provare il programma, abbiamo agito così:

- formattato un dischetto dati,
- lanciato il programma CREAREL,
- listato la directory del dischetto dati dopo l'esecuzione del programma, che riportiamo:

```
0 [illegible]
4     "PROVA REL"          REL
660 BLOCKS FREE.

READY.
```

puoi vedere che il file occupa 4 blocchi, ovviamente 3 per i dati e 1 per l'unico side sector necessario,

- abbiamo caricato il programmma DCOMEFS e l'abbiamo lanciato usando come dischetto quello con il file PROVA REL, abbiamo così osservato che il file inizia nel settore 17, 0 e che il side sector si trova nel settore 17,10, come puoi vedere nella parte di risultati sotto riportati:

```
RELATIVI  RR
 160  50  65  160  160  160  160


REL   17  0 PROVA REL
 17  10  21
 0  0
 4  0
DEL   0  0
 0  0  0
 0  0
 0  0
DEL   0  0
 0  0  0
 0  0
 0  0
```

• abbiamo caricato il programmma DISPLAY T&S, e abbiamo stampato i settori del dischetto dati che contengono il file PROVA REL, cioè il 17,0; il 17,11; il 17,1; e poi il 17,10.

Non riportiamo qui i risultati della stampa, che puoi facilmente ottenere anche tu. Ti consigliamo, però, di fare la prova, per renderti conto di quanto abbiamo esposto sui file relativi; vedrai, per esempio, i record a cavallo dei settori.

Segue il programma SCRIVIREL, con il quale si possono scrivere record nel file PROVA REL, prima creato. Questo programma scrive nei record logici voluti due campi dati, il primo di 10 caratteri e il secondo di 9, più i fine campo. Puoi scrivere il record logico che desideri; se dai due volte lo stesso numero di record, il secondo si sovrappone al primo.

```
5 REM SCRIVIREL
10 REM SCRITTURA RECORD NEL FILE RELATIVO
15 REM OGNI RECORD 2 CAMPI
20 REM NOME DI 10 CARATTERI
25 REM TELEFONO DI 9 CARATTERI
30 REM LUNGHEZZA RECORD=10+1+9+1=21
35 DR$="0":NF$=DR$+":PROVA REL,L,"
40 LU=21:SA=2:LF=2:B$="          "
45 BI=1:NR=30
50 OPEN15,8,15,"I"
55 REM APERTURA FILE
60 OPENLF,8,SA,NF$+CHR$(LU)
65 GOSUB160
70 PRINT"NOME=* PER USCIRE":PRINT
75 INPUT"NOME(10)";N$
80 IFN$="*"THEN155
85 N$=LEFT$(N$+B$,10)
90 INPUT"TEL. (9)";T$:T$=LEFT$(T$+B$,9)
95 INPUT"RECORD ";R
100 IFR=0ORR>NRTHENPRINTCHR$(145);:GOTO95
105 REM PREPARAZIONE PUNTATORE
110 HI=INT(R/256):LO=R-HI*256
115 C$="P"+CHR$(SA)
120 C$=C$+CHR$(LO)+CHR$(HI)
125 C$=C$+CHR$(BI)
130 REM PREDISPONE PUNTATORE PER SCRIVERE
```

```
135 PRINT#15,C$:GOSUB160
140 REM SCRIVE RECORD
145 PRINT#LF,N$;CHR$(13);T$:GOSUB160
150 GOTO70
155 CLOSELF:GOSUB160:CLOSE15:END
160 REM ROUTINE ERRORE, ACCETTA COD. 50
165 INPUT#15,EN,EM$,ET,ES
170 IFEN=0OREN=50THENRETURN
175 PRINTEN;EM$;ET;ES
180 STOP:RETURN
```

Il programma opera così:

- Prepara le costanti, linee 5-45.
- Apre il canale 15 e inizializza il dischetto dati, linea 50.
- Apre il file relativo PROVA REL; si potrebbe omettere L e la lunghezza, dato che il file esiste già. Linee 60-65.
- Chiede i dati per un record e il numero del record predispone il puntatore e scrive il record, linee 70-150.
- Se si risponde con "*" al nome, chiude il file e termina.

Dopo aver fatto girare il programma puoi andare a stampare il contenuto dei blocchi, come suggerito prima, con DISPLAY T&S; vedrai i record che hai caricato.

Segue il programma LEGGIREL per leggere un record qualunque dal file PROVA REL.

```
5 REM LEGGIREL
10 REM LETTURA RECORD
15 DR$="0":NF$=DR$+":PROVA REL,L,"
20 LU=21:SA=2:LF=2
25 BI=1:NR=30
30 OPEN15,8,15,"I"
35 REM APRE FILE
40 OPENLF,8,SA,NF$+CHR$(LU)
45 GOSUB135
50 PRINT"RECORD=0 PER USCIRE":PRINT
55 INPUT"RECORD ";R
60 IFR=0THEN130
```

```
65 IFR<0ORR>NRTHENPRINTCHR$(145);:GOTO55
70 REM PREPARA PUNTATORE
75 HI=INT(R/256):LO=R-HI*256
80 C$="P"+CHR$(SA)
85 C$=C$+CHR$(LO)+CHR$(HI)
90 C$=C$+CHR$(BI)
95 REM PREDISPONE PUNTATORE
100 PRINT#15,C$:GOSUB135
105 REM LEGGE RECORD
110 INPUT#LF,N$,T$:GOSUB135
115 PRINT"NOME: ";N$
120 PRINT"TEL.: ";T$
125 GOTO50
130 CLOSELF:GOSUB135:CLOSE15:END
135 REM ROUTINE ERRORE, AMMETTE COD. 50
140 INPUT#15,EN,EM$,ET,ES
145 IFEN=0OREN=50THENRETURN
150 PRINTEN;EM$;ET;ES
155 STOP:RETURN
```

Esso, dopo aver preparato le costanti e aperto il file, chiede il numero del record desiderato, calcola il puntatore, lo posiziona e va a leggere. Se vuoi leggere il file, come sequenziale, lo puoi fare con un solo posizionamento iniziale.

Segue il programma LEGGECAMPOREL, che invece di leggere un record, va a leggere il secondo campo di un record, dando un valore opportuno al puntatore.

```
5 REM LEGGICAMPOREL
10 REM LETTURA SOLO DEL SECONDO CAMPO
15 REM DI UN RECORD R
20 DR$="0":NF$=DR$+":PROVA REL,L,"
25 LU=21:SA=2:LF=2
30 BI=12:NR=30
35 INPUT"RECORD ";R
40 IFR=0ORR>NRTHENPRINTCHR$(145);:GOTO30
45 OPEN15,8,15,"I"
50 REM APREFILE
55 OPENLF,8,SA,NF$+CHR$(LU)
```

```
60 GOSUB120
65 REM PREPARA PUNTATORE
70 HI=INT(R/256):LO=R-HI*256
75 C$="P"+CHR$(SA)
80 C$=C$+CHR$(LO)+CHR$(HI)
85 C$=C$+CHR$(BI)
90 REM POSIZIONA PUNTATORE
95 PRINT#15,C$:GOSUB120
100 REM LEGGE CAMPO
105 INPUT#LF,T$:GOSUB120
110 PRINT"TEL.: ";T$
115 CLOSELF:GOSUB120:CLOSE15:END
120 REM ROUTINE ERRORE, AMMETTE COD. 50
125 INPUT#15,EN,EM$,ET,ES
130 IFEN=0OREN=50THENRETURN
135 PRINTEN;EM$;ET;ES
140 STOP:RETURN
```

Vogliamo farti riflettere sul fatto che il sistema, per consentirti di scrivere, per esempio, il quinto record del file, deve:

- determinare a quale settore appartiene il record,
- leggere il settore in un buffer,
- predisporre il puntatore nella giusta posizione,
- trasferire nel buffer i dati che tu scrivi,
- riscrivere sul dischetto l'intero settore.

Tutte queste operazioni per te risultano trasparenti, ma vengono eseguite. Nel caso dei file random, invece, queste operazioni le deve predisporre il programmatore nel suo programma.

La comodità di uso dei file relativi si paga con una maggiore occupazione di spazio disco, al massimo i sei settori dedicati ai side sector.

Anche per i file relativi sussiste il problema del reperimento del record in base ad un campo chiave ed è necessario ricorrere ad un file indice sequenziale, come per i file random. Solo che, in questo caso, nell'indice deve comparire la chiave del record, e, come indirizzo, il suo numero d'ordine nel file.

4.2 ESEMPIO DI ARCHIVIO RELATIVO

Abbiamo modificato il programma ARCHIRANDOM per costruire il programma ARCHIRELATIVO.
Anche in questo caso abbiamo creato, in fase di generazione del file, un file indice sequenziale INDI1, avente come chiave di ordinamento i primi due campi del record logico: cognome e nome. L'indirizzo del record è il suo numero d'ordine nel file, in conseguenza il record del file indice INDI1 è formato solo da due campi, chiave e numero d'ordine.
È possibile costruire un indice secondario INDI2, con le stesse caratteristiche di quello dei programmi esempio precedenti.

Nella fase di creazione dell'archivio ex-novo, viene preesteso il file relativo al numero dei record richiesto, e viene creato il file indice INDI1. Il file indice INDI1 deve sempre essere riscritto tutto sul dischetto, per non perdere indirizzi. Questa volta abbiamo modificato il modo di gestire i dati generali dell'archivio; abbiamo creato un piccolo file sequenziale, di nome DATIGEN, che contiene il nome del file relativo, la data di aggiornamento, il numero totale di record previsti e il numero di record in essere. Il file DATIGEN viene letto all'inizio dell'elaborazione e riscritto alla fine.

```
5 REM ARCHIRELATIVO
10 REM-----------------------------------
15 REM DEFINIZIONE COSTANTI E VARIABILI
20 REM-----------------------------------
25 NC=14:REM **NUMERO CAMPI RECORD**
30 REM       **********************
35 S1$="DATA ULTIMO AGG.       "
40 S2$="FINITO SPAZIO ASSEGNATO"
45 SP$="                        "
46 SWW=0:REM PER EVITARE RIDIMENSIONAMENTO
50 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(99)+CHR$(99)
55 DIMD$(NC):DIMY$(NC):DIML(NC)
60 DATA"COGNOME:","NOME    :","INDIR.  :"
65 DATA"CITTA'  :","PROV.   :","TELEF.  :"
70 DATA"L.NASC.:","D.NASC.:","T.STUD.:"
75 DATA"OCC.AT.:","OCC.PR.:","ST.CIV.:"
80 DATA"NOTA 1  :","NOTA 2  :"
85 DATA20,15,30,25,11,10,20,8,20,20,20,1,20,20
90 FORJ=1TONC:READD$(J):NEXTJ
95 FORJ=1TONC:READL(J):NEXTJ
```

```
100 REM **PRESENTAZIONE MENU' E SCELTE**
105 REM ********************************
110 CLOSE15:OPEN15,8,15
115 PRINT"□□□";TAB(10);"GESTIONE ARCHIVIO":PRINT
120 PRINTTAB(10)"1=INIZIO EX-NOVO"
125 PRINTTAB(10)"2=AGGIORNAMENTO"
130 PRINTTAB(10)"3=LISTA PRINCIPALE"
135 PRINTTAB(10)"4=CREAZ.IND.SEC."
140 PRINTTAB(10)"5=LISTA SECONDARIA"
145 PRINTTAB(10)"9=FINE"
150 INPUT"COSA SCEGLI ";X
155 IFX<1ORX>5ANDX<>9THENPRINT"□":GOTO150
160 PRINT:GOSUB675
165 R$="":PRINTTAB(10)"MONTA DISCO DATI"
170 GETR$:IFR$=""THEN170
175 IFX=1THEN210
180 PRINT#15,"I"
185 GOSUB580:PRINT"PRESENTI ";K;" RECORD"
186 NN$=N$+",L,"+CHR$(254)
190 PRINTS1$;G1$;"/";M1$;"/";A1$
191 R$="":GETR$:IFR$=""THEN191
192 IFSWW=0THENDIMC$(N),T%(N)
193 GOSUB615
195 IFX=9THEN1340
200 ONX-1GOTO825,1150,1235,1290
205 STOP
210 REM-------------------------
215 REM INIZIO EX-NOVO ARCHIVIO
220 REM-------------------------
225 PRINT"POSSO FORMATTARE IL DISCO DATI S/N"
230 R$="":INPUTR$:IFR$=""THEN230
235 IFR$="S"THEN240
237 STOP
240 GOSUB740
250 INPUT"NOME FILE DA INIZIARE: ";N$
255 INPUT"QUANTI RECORD IN TUTTO: ";N
260 DIMC$(N),T%(N):SWW=1
261 REM PREPARAZIONE CAMPI DUMMY
263 P1$="*":FORJ=1TO52:P1$=P1$+" ":NEXTJ
265 P2$=LEFT$(P1$,37)
273 REM PREESTENSIONE FILE N$
274 OPEN11,8,11,N$+",L,"+CHR$(254)
```

```
275 FORJ=NTO1STEP-1
280 HI=INT(J/256):LO=J-HI*256
285 C$="P"+CHR$(11)+CHR$(LO)+CHR$(HI)+CHR$(1)
290 PRINT#15,C$:GOSUB645
293 PRINT#11,P1$CH$P1$CH$P1$CH$P1$CH$P2$CH$;
294 GOSUB645
295 NEXTJ:CLOSE11:GOSUB645
300 REM PREPARAZIONE INDICE
305 FORJ=1TON:T%(J)=J
310 C$(J)=LIM$:NEXTJ
315 GOSUB390:REM SCRIVE INDI1
320 K=0:GOSUB485:GOTO110
325 REM **INGRESSO DATI**
330 REM ******************
335 PRINT"INGRESSO DATI"
340 FORJ=1TONC:Y$(J)="":NEXTJ
345 PRINT"PER USCIRE $ PER COGNOME";
350 PRINT"SE MANCANO DATIPREMI SOLO RETURN"
355 PRINTD$(1);:INPUTY$(1)
360 IFY$(1)="$"THENW=1:RETURN
365 FORJ=2TONC:PRINTD$(J);:INPUTY$(J):NEXTJ
370 REM **SISTEMA LUNGHEZZA DATI**
375 REM ****************************
380 FORJ=1TONC:Y$(J)=LEFT$(Y$(J)+SP$,L(J))
385 NEXTJ:GOSUB1020:RETURN
390 REM **SCRITTURA INDICE PRINCIPALE**
395 REM *******************************
400 PRINT#15,"S0:INDI1":PRINT#15,"I"
405 OPEN10,8,10,"INDI1,S,W":GOSUB645
410 FORJ=1TON
415 PRINT#10,C$(J);CH$;T%(J);CH$;
420 GOSUB645:NEXTJ
425 CLOSE10:RETURN
430 REM **SCRITTURA RECORD PUNTATO**
435 REM ****************************
437 REM **COSTRUZIONE STRINGA DI STAMPA**
438 REM ********************************
440 C$="":FORJ=1TONC
441 Y$(J)=LEFT$(Y$(J)+SP$,L(J))
443 C$=C$+Y$(J)+CH$:NEXTJ
445 PRINT#11,C$;:GOSUB645
450 RETURN
```

```
455 REM **DEFINISCE POSIZIONE RECORD K**
460 REM ********************************
465 HI=INT(T/256)
470 LO=T-HI*256
475 PRINT#15,"P"+CHR$(11)+CHR$(LO)+CHR$(HI)+CHR$(1)
480 GOSUB645:RETURN
485 REM **SCRITTURA DATIGEN DISCO**
490 REM ***************************
495 PRINT#15,"S0:DATIGEN"
500 PRINT#15,"I"
505 OPEN10,8,10,"DATIGEN,S,W":GOSUB645
510 PRINT#10,N$CH$G$M$A$CH$;N;CH$;K;CH$;
515 CLOSE10:RETURN
520 REM **RICERCA POSTO NELL'INDICE**
525 REM ****************************
530 JJ=0:FORJ=1TON
535 IFLEFT$(C$(J),1)=CHR$(99)THEN 545
540 NEXTJ:RETURN
545 JJ=J:J=N:GOTO540
550 REM **LETTURA RECORD**
555 REM ******************
560 PRINT#15,"I":OPEN11,8,11,NN$:GOSUB645
565 GOSUB455
570 FORJ=1TONC:INPUT#11,Y$(J):GOSUB645
575 NEXTJ:CLOSE11:RETURN
580 REM **LETT.DATI DISCO**
585 REM ******************
590 OPEN10,8,10,"DATIGEN,S,R":GOSUB645
595 INPUT#10,N$,R$,N,K:GOSUB645
600 CLOSE10:GOSUB645
605 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2)
610 A1$=RIGHT$(R$,2):RETURN
615 REM **LETT. INDICE**
620 REM ****************
625 OPEN10,8,10,"INDI1,S,R":GOSUB645
630 FORJ=1TON
635 INPUT#10,C$(J),T%(J):GOSUB645
640 NEXTJ:CLOSE10:RETURN
645 REM **ROUTINE ERRORE**
650 REM ******************
655 INPUT#15,EN,EM$,ET,ES
660 IFEN=0OREN=50THENRETURN
```

```
665 PRINT"ERRORE DISCO"
670 PRINTEN,EM$,ET,ES:CLOSE15:STOP
675 REM **DATA DISCO**
680 REM **************
685 PRINT"DATA PER DISCO"
690 INPUT"  GG,MM,AA            ";G$,M$,A$:RETURN
695 REM **SCRITT. IND. SEC.**
700 REM *********************
705 PRINT#15,"S:INDI2":PRINT#15,"I"
710 OPEN10,8,10,"INDI2,S,W"
715 FORJ=1TOK
720 PRINT#10,C$(J);CH$;T%(J);CH$;
725 GOSUB645:NEXTJ:CLOSE10:RETURN
730 R$="":GETR$:IFR$=""THEN730
735 RETURN
740 REM-------------------------
745 REM INIZIALIZZAZIONE DISCO
750 REM-------------------------
755 PRINT"NOME DISCO";:INPUTN$
760 PRINT#15,"N0:"+N$+",99"
765 CLOSE15:OPEN15,8,15:PRINT#15,"I"
770 RETURN
775 REM **NUOVI DATI**
776 REM **************
777 OPEN11,8,11,NN$:GOSUB645
779 GOSUB325
780 IFW=1THENK=K-1:CLOSE11:GOTO1340
785 GOSUB520:IFJJ=0THENSTOP:REM STOP IMPOSS.
790 T=T%(JJ):GOSUB455:GOSUB430
795 REM **AGGIORNAMENTO INDICE**
800 REM ************************
805 C$(JJ)=Y$(1)+Y$(2)
810 K=K+1
815 IFK>NTHENK=K-1:PRINTS2$:CLOSE11:GOTO1340
820 GOTO779
825 REM----------------
830 REM AGGIORNAMENTO
835 REM----------------
840 PRINTTAB(10);"AGGIORNAMENTO ARCHIVIO"
845 PRINT:PRINTTAB(10);"1=CORREZIONE"
850 PRINTTAB(10);"2=AGGIUNTA ELEM."
855 PRINTTAB(10);"3=CANCELL.ELEM."
```

```
860 PRINTTAB(10);"9=FINE"
865 INPUT"COSA SCEGLI ";X:IFX=9THEN1340
870 IFX<1ORX>3THEN840
875 IFX=2THEN1070
880 IFX=3THEN1110
885 GOTO970
890 REM **RICERCA RECORD**
895 REM ******************
900 PRINT"□";D$(1);:INPUTY$(1):W=0
905 IFY$(1)="$"THENW=1:RETURN
910 PRINTD$(2);:INPUTY$(2)
915 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
920 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
925 INPUT"QUALE OCCORRENZA ";X
930 IFX<=0THENPRINT"□";:GOTO925
935 FORJ=1TOK:IFC$(J)=Y$(1)+Y$(2)THEN955
940 NEXTJ
945 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
950 GOSUB730:GOTO890
955 IFX<>1THENX=X-1:GOTO940
960 T=T%(J)
965 I=J:J=K:NEXTJ:RETURN
970 REM **CORREZIONE**
975 REM ***************
980 GOSUB890:IFW=1THEN840
985 GOSUB550
990 REM **VA A MODIFICA DATI**
995 REM **********************
1000 GOSUB1020
1005 PRINT#15,"I":OPEN11,8,11,NN$
1006 GOSUB455:GOSUB430
1010 C$(I)=Y$(1)+Y$(2)
1015 CLOSE11:GOTO970
1020 REM **CONTROLLO DATI E MODIFICHE**
1025 REM ******************************
1030 PRINT"□CONTROLLO DATI E MODIFICHE"
1035 FORJ=1TONC:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
1040 INPUT"□TUTTO BENE S/N ";X$:IFX$="S"THENRETURN
1045 PRINT"QUALE CAMPO (0 USCITA)";:INPUTX
1050 IFX=0THENRETURN
1055 IFX>NCORX<1THENPRINT"□":GOTO1040
1060 Y$(X)=""
```

```
1063 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
1065 GOTO1030
1070 REM **AGGIUNTA**
1075 REM ************
1085 PRINT"□AGGIUNTA NUOVI RECORD"
1090 PRINT"PRESENTI ";K;" RECORD"
1095 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD"
1100 GOSUB730
1105 W=0:K=K+1:GOTO775
1110 REM **CANCELLAZIONE**
1115 REM ****************
1120 M=0
1125 GOSUB890:IFW=1THEN1140
1130 X=L(1)+L(2):C$(I)=LEFT$(LIM$+SP$+SP$,X)
1135 M=M+1:GOTO1125
1140 REM SIST. INDICE
1145 GOSUB1385:K=K-M:GOTO1365
1150 REM-------------
1155 REM LISTA FILE
1160 REM-------------
1165 T$="LISTA PER INDICE PRIM."
1170 PRINT"ACCENDI STAMPANTE     PREMI UN TASTO"
1175 GETR$:IFR$=""THEN1175
1180 PRINT"□";T$:OPEN4,4
1185 PRINT#4:PRINT#4:PRINT#4,T$
1190 FORJ=1TO10:PRINT#4:NEXTJ
1195 L=2:FORI=1TOK:T=T%(I):GOSUB550
1200 PRINT#4,Y$(1);"  ";Y$(2)
1205 PRINT#4,Y$(3);"  ";Y$(4);"  ";Y$(5)
1210 FORM=6TONC:PRINT#4,D$(M);Y$(M):NEXTM
1215 PRINT#4:PRINT#4
1220 IFL=5THENPRINT#4:L=0
1225 L=L+1:NEXTI
1230 CLOSE4:GOTO1370
1235 REM--------------------------------
1240 REM CREAZIONE INDICE SECONDARIO
1245 REM--------------------------------
1250 PRINT"□INDICE SECONDARIO"
1255 INPUT"■QUALE CAMPO ";X$
1260 X=VAL(X$):IFX<2ORX>NCTHENGOTO1255
1265 FORI=1TOK:T=T%(I):GOSUB550
1266 IFLEN(Y$(X))=0THENY$(X)=CHR$(160)
```

```
1270 C$(I)=Y$(X):NEXTI
1275 GOSUB695:GOSUB615
1280 PRINT"FINITO IND. SEC."
1285 GOTO1370
1290 REM-------------------------------
1295 REM LISTA PER INDICE SECONDARIO
1300 REM-------------------------------
1305 PRINT"LISTA IND. SEC."
1310 OPEN10,8,10,"INDI2,S,R":GOSUB645
1315 FORJ=1TOK:INPUT#10,C$(J),T%(J)
1320 GOSUB645:NEXTJ
1325 CLOSE10:GOSUB1450:GOSUB695
1330 T$="LISTA IND. SEC. "
1335 GOTO1170
1340 REM **CHIUSURA**
1345 REM ************
1350 PRINT"CHIUSURA ARCHIVIO"
1355 PRINT"IN ORDINA INDICE INDI1"
1356 PRINT" ATTENDI CON PAZIENZA"
1360 GOSUB1385
1365 GOSUB390:GOSUB485
1370 PRINT"FINITO AGGIORNAMENTO"
1375 PRINT"SONO PRESENTI ";K;" RECORD"
1380 CLOSE15:STOP
1385 REM **ORDINAMENTO CRESCENTE**
1390 REM ************************
1395 L=N-1
1400 W=0
1405 FORJ=1TOL
1410 IFC$(J)<=C$(J+1)THEN1430
1415 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
1420 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
1425 W=1
1430 NEXTJ
1435 IFW=0THENRETURN
1440 IFL=1THENRETURN
1445 L=L-1:GOTO1400
1450 REM **ORDINAMENTO DECRESCENTE**
1455 REM ************************
1460 L=K-1
1465 W=0
1470 FORJ=1TOL
```

```
1475 IFC$(J)>=C$(J+1)THEN1495
1480 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
1485 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
1490 W=1
1495 NEXTJ
1500 IFW=0THENRETURN
1505 IFL=1THENRETURN
1510 L=L-1:GOTO1465
```

Non riteniamo di dover descrivere in dettaglio anche questo programma, ma ci limitiamo a segnalare i blocchi significativi per il tipo di file relativo.

SCRITTURA RECORD PUNTATO
linee 430-450
Viene eseguito dopo il blocco: DEFINISCE POSIZIONE RECORD (linee 455-480), che prepara il puntatore all'inizio di un determinato record. Come vedi prepariamo una stringa C$, che contiene la somma di tutte le stringhe e i caratteri separatori necessari per riempire il record. L'istruzione PRINT#11,C$, scrive tutto il record logico.

INIZIO EX-NOVO ARCHIVIO
linee 210-320
Dopo aver formattato il dischetto, viene preesteso il file, generato il primo file indice INDI1 e generato il primo file DATIGEN.
Ti segnaliamo l'uso dello switch SWW per evitare il ridimensionamento dei vettori, linee 260 e 192.

RICERCA POSTO NELL'INDICE
linee 520-540
Serve per trovare il numero del record per un nuovo record da aggiungere.

Pensiamo che, per una completa comprensione delle operazioni possibili con i file su disco, sia molto utile studiare i quattro programmi di archivio presentati. Essi, ovviamente, potrebbero essere migliorati sotto diversi aspetti. Il più evidente è il possibile e facile miglioramento dei quadri video, facendo ricorso anche al colore; noi non ce ne siamo occupati, dato che il nostro argomento principale è di spiegare come si gestiscono i file. Lasciamo a te la cura e la soddisfazione di proseguire e migliorare il nostro lavoro. Vedi quanto detto in proposito nel primo volume.

Noi abbiamo provato il programa ARCHIRELATIVO generando il file ARCHI-REL di 500 record logici. Riportiamo la directory del dischetto usato.

```
0 [illegible]
505   "ARCHIREL"          REL
20    "INDI1"             SEQ
1     "DATIGEN"           SEQ
138 BLOCKS FREE.
```

Come puoi vedere, il file relativo occupa 505 settori, 500 per i record logici e 5 per i side sector necessari. Il file sequenziale DATIGEN occupa un solo settore. Il file INDI1 occupa solo 20 settori, infatti nella fase di impostazione le chiavi DUMMY sono solo di 3 caratteri; man mano che si aggiungono i record, le dimensioni di INDI1 crescono, ma non si arriverà mai ai 90 settori usati per l'esempio dell'archivio RANDOM, dato che in questo caso il record del file indice ha un campo meno.

CAPITOLO 5

ERRORI

5.1 MESSAGGI DI ERRORE DEL DOS

L'indicatore luminoso a luce rossa, posto sull'unità 1541, si accende quando è in corso una operazione di lettura o scrittura da o sul dischetto. Quando si verifica un errore, l'indicatore pulsa velocemente. Come abbiamo visto più volte, la routine di errore, richiedendo il contenuto di 4 variabili sul canale 15 può rilevare se si è verificato un errore, e, nello stesso tempo resetta (spegne) l'indicatore.
Le 4 variabili, che possono essere tutte di tipo stringa, ma, salvo la seconda, anche di tipo numerico, e che noi abitualmente chiamiamo EN,EM$,ET,ES, contengono:
EN, il numero dell'errore o del messaggio di avvertimento,
EM$, il messaggio, se esiste,
ET, la traccia coinvolta,
ES, il settore coinvolto.

Il messaggio di codice 0, significa che è andato tutto bene, mentre quello di codice 1, segnala il numero di file cancellati in una operazione di SCRATCH.

Riportiamo l'elenco degli altri messaggi.

20 READ ERROR
Il controller del disco non riesce a trovare la testata (header) del settore richiesto; o l'indirizzo passato non è valido, o il dischetto è rovinato.

21 READ ERROR
Il controller del disco non trova il carattere di sincronizzazione sulla traccia richiesta. Può essere allineata male la testina di lettura/scrittura, mancare, essere inserito male o non essere formattato il dischetto. Può trattarsi di un guasto hardware.

22 READ ERROR
È stata richiesta una operazione di tipo BLOCK, ma non può essere eseguita, o perchè il blocco è danneggiato o perchè l'indirizzo non è valido.

23 READ ERROR
Dopo la lettura di un blocco il controllo di CHECKSUM dà errore. Può dipendere da una difettosa messa a terra.

24 READ ERROR
Viene provocato un errore hardware a causa di una cattiva ricezione dei dati o di una testata. Può dipendere da una difettosa messa a terra.

25 WRITE ERROR
I dati registrati e i dati in memoria non coincidono.

26 WRITE PROTECT ON
Si tenta di scrivere su un dischetto protetto, con la finestra chiusa.

27 READ ERROR
Il controller del disco ha trovato un errore nella lettura di una testata e non ha letto il blocco. Può dipendere da una difettosa messa a terra.

28 WRITE ERROR
Il controllore cerca, dopo aver scritto un blocco, il carattere di sincronizzazione del blocco seguente e non lo trova entro il tempo prefissato. Il dischetto può essere rovinato o formattato male, o si è verificato un guasto hardware.

29 DISK ID MISMATCH
Indica il tentativo di accedere a un dischetto che non è stato inizializzato o che è rovinato.

30 SYNTAX ERROR
Il DOS non riesce a interpretare un comando ricevuto; probabilmente sono errati alcuni parametri o ne mancano.

31 SYNTAX ERROR
Il DOS non riconosce un comando; può dipendere da uno spazio prima del comando.

32 SYNTAX ERROR
Il comando inviato supera la massima lunghezza consentita.

33 SYNTAX ERROR
Il nome usato in OPEN o SAVE non è valido.

34 SYNTAX ERROR
Il DOS non riconosce il nome del file; per esempio, mancano i due punti dopo il numero dell'unità.

39 SYNTAX ERROR
Il comando inviato sul canale 15 non viene riconosciuto.

50 RECORD NOT PRESENT
Si tenta di leggere da un file dopo aver raggiunto la fine del file. Nel caso dei file relativi può essere ignorato se si sta estendendo il file.

51 OVERFLOW IN RECORD
Si tenta di scrivere in un blocco più caratteri di quelli possibili. Per esempio, non si è tenuto conto dei caratteri separatori tra i campi.

52 FILE TOO LARGE
Si scrivono record su un file relativo producendo overflow.

60 WRITE FILE OPEN
Si apre in lettura un file che è stato scritto e non chiuso.

61 FILE NOT OPEN
Si tenta un'operazione su un file che non è stato aperto. In alcuni casi può non comparire il messaggio, però l'operazione non viene eseguita.

62 FILE NOT FOUND
Il file richiesto non esiste.

63 FILE EXISTS
Si cerca di creare un file che esiste già e non si è usato il carattere "@" prima del nome.

64 FILE TYPE MISMATCH
Il tipo di operazione si riferisce a un file di tipo diverso da quello registrato.

65 NO BLOCK
Indica che il blocco richiesto con B-A è già occupato, ma fornisce in ET e ES il numero di traccia e settore del primo bblocco disponibile. Se ET e ES sono nulli,

significa che il dischetto è pieno.

66 ILLEGAL TRACK AND SECTOR
Il DOS tenta di accedere a settori che non esistono.

67 ILLEGAL SYSTEM T OR S
Indica una traccia e un settore illegali per il sistema.

70 NO CHANNEL
Indica che il canale richiesto è già occupato o che non ci sono canali liberi.

71 DIRECTORY ERROR
Indica che i controlli sul dischetto non permettono di creare una BAM valida. Probabilmente si deve perdere il contenuto e riformattare il dischetto.

72 DISK FULL
Indica o che il disco è pieno, o che non ci sono più entrate libere nella directory.

73 DOS MISMATCH (73, CBM DOS V2.6 1541) Le versioni 1 e 2 del DOS creano dischetti compatibili in lettura, ma non in scrittura. Si cerca di scrivere su un dischetto formattato con l'altra versione.

74 DRIVE NOT READY
Si tenta di lavorare sull'unità spenta.

Non tutti i messaggi di errore vengono segnalati spontaneamente dal sistema. Per questo ti raccomandiamo di usare sempre la routine di errore nei tuoi programmi.

CAPITOLO 6

PROGRAMMI DI UTILITA'

6.1 PROGRAMMI DI UTILITA'

I programmi di utilità sono di aiuto durante il lavoro di programmazione. Nella scatola dell'unità 1541 è contenuto un dischetto TEST/DEMO che contiene alcuni programmi di utilità e alcuni programmi esempio. Riportiamo la directory del TEST/DEMO.

```
0 [illegible]
13    "HOW TO USE"        PRG
5     "HOW PART TWO"      PRG
4     "VIC-20 WEDGE"      PRG
1     "C-64 WEDGE"        PRG
4     "DOS 5.1"           PRG
```

```
11    "COPY/ALL"          PRG
9     "PRINTER TEST"      PRG
4     "DISK ADDR CHANGE"  PRG
4     "DIR"               PRG
6     "VIEW BAM"          PRG
4     "CHECK DISK"        PRG
14    "DISPLAY T&S"       PRG
9     "PERFORMANCE TEST"  PRG
5     "SEQUENTIAL FILE"   PRG
13    "RANDOM FIAL"       PRG
558 BLOCKS FREE.
```

I primi due programmi: HOW TO USE e HOW PART TWO, mostrano sul video le spiegazioni relative agli altri programmi del dischetto. Puoi caricare il primo, dare RUN e seguire le istruzioni che compaiono sul video. Al termine del primo viene caricato automaticamente il secondo.

Il programma C-64 WEDGE serve per caricare il DOS 5.1, se non è già stato caricato prima. Dopo aver dato RUN, vedi comparire READY, e sono attivi i seguenti nuovi comandi:

/nome-programma, per caricare un programma da dischetto;

> oppure @, per mostrare lo stato delle 4 variabili di errore;

> $ oppure @$, per vedere la directory sul video, SENZA AZZERARE IL PROGRAMMA PRESENTE IN MEMORIA.
Il programma resta attivo fino a quando togli corrente al calcolatore.

Il programma COPY/ALL consente di fare la copia dei dischetti usando due unità 1541 collegate al calcolatore, una con numero 8 e l'altra con numero 9. Per poter cambiare a una delle unità il numero da 8 a 9, puoi usare il programma DISK ADDR CHANGE, seguendo le istruzioni che compaiono sul video. Ricorda che il cambio del numero dell'unità resta valido fino a quando togli corrente. Per ottenere una unità che risponda sempre al numero 9, si deve modificare via hardware il numero, operando all'interno.
Ti consigliamo, se usi COPY/ALL, o altro programma di copia (ne esistono in commercio, che lavorano con una sola unità collegata), di proteggere la finestra del disco sorgente per lavorare senza pericoli. Usando due unità per copiare dischetti, l'operazione risulta più veloce. Con una sola unità, si devono alternare i dischetti alcune volte, fino al compimento dell'operazione.

Il programma PRINTER TEST consente di provare le caratteristiche della stampante; ti conviene usarlo, almeno una volta, per renderti conto di quali possibilità di stampa disponi.

Il programma DIR è abbastanza interessante anche per le tecniche di programmazione usate. Puoi listarlo sulla stampante o sul video e seguire il listato leggendo il nostro commento.
Viene mostrato il MENU':

D-DIRECTORY
>-DISK COMMAND
Q-QUIT PROGRAM
S-DISK STATUS

per la scelta. Q fa uscire dal programma.
Il canale comandi viene aperto con lfn=2 all'inizio. Scegliendo D, il programma va alla linea 10 e apre la DIRECTORY come file "$0", sul canale 0, con lfn=1. Poi con GET#1, viene letta e stampata la directory carattere per carattere. Questo è un modo diverso dall'abituale per ottenere la stampa della directory.
Rispondendo S, viene letto, in modo diverso dal solito, lo stato delle 4 variabili d'errore. Vengono eseguite sul canale comandi, lfn=2, delle GET#2, fino ad incontrare RETURN, e si stampano i caratteri. Trovi questo alle linee 5000-5020.
Rispondendo >, il programma lo stampa come PROMPT e si mette in attesa di una stringa comando DOS; quando premi RETURN, viene eseguito il tuo comando con la PRINT#2 della linea 4020.

Il programma VIEW BAM mostra sul video la BAM del dischetto, dapprima le tracce da 1 a 17 e poi le tracce da 18 a 35. La BAM viene vista come una matrice, sull'asse orizzontale sono riportate le tracce e sull'asse verticale i settori. Le posizioni scure della matrice indicano i blocchi occupati. Il programma presenta una imperfezione, per le tracce da 18 a 24 sono mostrati, senza l'annullamento N/L, ma come occupati, anche i settori numero 19, che non sono disponibili.

Il programma CHECK DISK consente di controllare se un dischetto ha settori rovinati, dove non si riesce a scrivere. All'inizio il programma esegue la VALIDATE, quindi cancella i settori non registrati nella directory. Dopo il programma scrive nei settori liberi, usando le funzioni dirette del DOS, e alloca i settori dove scrive; tali settori non sono però registrati nella directory. Se non riesce a scrivere in un settore, ne annota l'indirizzo in una doppia tabella di 100 elementi. Alla fine segnala gli indirizzi dei settori guasti. In realtà il dischetto alla fine risulta tutto occupato nella BAM, settori buoni e non buoni, senza corrispondenza con la directory. Per poter usare il dischetto si deve rifare la VALIDATE e andare ad

occupare stabilmente, sia nella BAM che nella Directory, i settori guasti. Per occupare la BAM basta usare il comando B-A, per occcupare la directory si può andare a scriverci dentro direttamente, usando il comando B-P e U2. Il programma risulta molto lento, e questo dipende da come vengono ricercati i settori liberi su cui andare a scrivere.

Il programma DISPLAY T&S è molto utile per controllare cosa succede sul disco. Noi, come hai potuto vedere leggendo questo capitolo, lo usiamo molto. Esso consente di visualizzare, o sul video o sulla stampante, il contenuto di un settore carattere per carattere. La visualizzazione avviene in due modi, con i codici esadecimali dei caratteri e con i caratteri di stampa. La parte in codice esadecimale, operando la conversione decimale, fornisce i codici ASCII abituali, la parte in caratteri di stampa risulta poco chiara, dato che alcuni codici non danno luogo a caratteri stampabili. Il programma in certi casi va in crisi. Basta premere RUN-/STOP e RESTORE e poi RUN per ripartire.

Il programma PERFORMANCE TEST serve per provare il buon funzionamento del dischetto.

Gli ultimi due programmi: SEQUENTIAL FILE e RANDOM FIAL, sono due esempi di creazione e gestione di file sequenziale e random.

Noi abbiamo preparato qualche altro programma di utilità che riportiamo qui.

Il programma DCOMEFS, stampa la directory, leggendola come file sequenziale, carattere per carattere. La stampa viene predisposta per evidenziare i contenuti della directory. La prima parte della stampa mostra i 35 gruppi (uno per ogni traccia) di 4 byte dedicati alla BAM; il primo byte di ogni quartina segnala il totale dei blocchi liberi nella traccia, gli altri sono usati in sequenza per il primo gruppo di 8 settori, per il secondo gruppo di 8 settori e per i settori restanti. Nella seconda parte della stampa sono evidenziate le altre informazioni; per i file nell'ordine: tipo file, indirizzo primo blocco file, nome file, tre byte significativi per i file REL (indirizzo del primo blocco side sector e lunghezza record), sono saltati 4 byte non usati, traccia e settore sostitutivi per uso di OPEN@, numero dei blocchi del file in 2 byte.

```
5 REM DCOMEFS
10 T$(0)="DEL":T$(1)="SEQ":T$(2)="PRG"
15 T$(3)="USR":T$(4)="REL"
20 CLOSE15:OPEN15,8,15,"I"
25 OPEN2,8,2,"$":REM APERTURA FILE DIRECTORY
30 OPEN4,4:REM APERTURA STAMPANTE
35 PRINT#4,"STAMPA DIRECTORY COME FILE SEQUENZIALE"
40 PRINT#4
```

```
45 REM LETTURA PRIMI 2 BYTE
50 I=2:GOSUB230
55 REM LETTURA BAM
60 FORK=1 TO 35:PRINT#4,K;"   ";
65 I=4:GOSUB230
70 NEXTK:PRINT#4
75 REM LETTURA NOME DISCO
80 I=18:GOSUB205
85 PRINT#4,N$;"  ";
90 REM LETTURA ID
95 I=2:GOSUB205:PRINT#4,N$
100 REM LETTURA ALTRI 7 BYTE
105 I=7:GOSUB230
110 PRINT#4
115 REM LETTURA ALTRI 85 BYTE
120 FORK=1TO85:GET#2,A$:NEXTK
125 PRINT#4
130 FORJ=1TO8
135 GET#2,T$,A$,B$
140 IFSTTHENCLOSE2:CLOSE4:CLOSE15:STOP
145 IFT$=""THENT$=CHR$(128)
150 I=16
155 GOSUB205
160 PRINT#4,T$(ASC(T$)-128);"  ";
165 PRINT#4,ASC(A$+CHR$(0));ASC(B$+CHR$(0));
170 PRINT#4,N$
175 I=3:GOSUB230
180 FORK=1TO4:GET#2,A$:NEXTK
185 I=2:GOSUB230
190 I=2:GOSUB230
195 IFJ<8THENGET#2,A$,A$
200 NEXTJ:GOTO130
205 REM COSTRUZIONE STRINGA
210 N$="":FORL=1TOI
215 GET#2,L$
220 IFL$<>CHR$(96)THENIFL$<>CHR$(160)THENN$=N$+L$
225 NEXTL:RETURN
230 REM LETTURA E STAMPA GRUPPO BYTE
235 FORL=1TOI
240 GET#2,A$:PRINT#4,ASC(A$+CHR$(0));
245 NEXTL:PRINT#4
250 RETURN
```

Non riportiamo i risultati che puoi ottenere tu. Se vuoi, puoi modificare la linea 30 per trasferire i risultati sul video, così: OPEN4,3. Seguendo le indicazioni sulla struttura della directory, puoi costruire un programma che ti permetta anche di modificarla, andando a scriverci dentro in modo diretto, con i comandi del DOS.

Segue il programma CONTRDISCO; esso controlla che i file di tipo SEQ e di tipo PRG siano registrati bene andando a confrontare la somma dei blocchi concatenati con il numero totale dei blocchi riportato nella directory. Salta nel controllo i file di altro tipo. Ogni volta che controlla un blocco, fa apparire un pallino sul video. Qualora per un file controllato vengano riscontrate irregolarità, il programma chiede se si vuole cancellarlo. Se la risposta è S, il file viene cancellato e viene eseguita la VALIDATE del dischetto; se la risposta è N il programma si ferma, dopo aver chiuso tutti i file.

```
5 REM CONTRDISCO
10 REM --COSTANTI E VARIABILI--
15 M1$="**************":FI=0
20 M2$="▒FILE NON CORRETTO: ▒"
25 M3$="▒TUTTO IN ORDINE   ...▒ SPERO !"
30 REM --APRE CANALE 15 CON LFN=1--
35 OPEN1,8,15,"I"
40 REM --APRE CANALE 2 PER DIRECTORY--
45 OPEN2,8,2,"#"
50 REM --APRE CANALE 3 PER FILE--
55 OPEN3,8,3,"#"
60 REM --FUNZ. PER PUNTARE ENTRATE DIRECTORY--
65 DEFFNF(X)=2+32*X
70 REM --PRIMO BLOCCO DIRECTORY--
75 TD=18:SD=1:GOSUB350
80 REM --ELABORA UNA ENTRATA--
85 FI$="":CB=0
90 REM --PUNTATORE A INIZIO ENTRATA--
95 PRINT#1,"B-P:2,"FNF(FI)
100 REM --LEGGE TIPO FILE CHE VA IN TY--
105 GET#2,A$:GOSUB385:TY=ASC(A$)
110 REM --LEGGE IND. INIZIO FILE, TR E SC--
115 GET#2,A$,B$:GOSUB380:TR=ASC(A$):SC=ASC(B$)
120 REM --LEGGE NOME FILE--
125 FORI=1TO16:GET#2,A$:FI$=FI$+A$
```

```
130 IFA$=CHR$(160)THENFI$=LEFT$(FI$,I-1):I=16
135 NEXT
140 REM --PUNTATORE AI 2 BYTE CONT. BLOCCHI--
145 PRINT#1,"B-P:2,"FNF(FI)+28:GET#2,A$,B$
150 REM --NB=VCONTATORE BLOCCHI--
155 GOSUB380:NB=ASC(A$)+ASC(B$)*256
160 REM --CONTROLLO TIPO FILE 1 O 2--
165 IFTY<>0THENTY=TY-128
170 IFTY<>2ANDTY<>1THEN310
175 IFTY<>2ANDTY<>1THEN85
180 REM --CONTROLLO BLOCCHI FILE--
185 PRINT"□□□"M1$
190 PRINT"CONTROLLO FILE":PRINTM1$"□□□"
195 PRINTTAB(10)FI$"□□"
200 REM --LEGGE BLOCCO FILE--
205 PRINT#1,"B-R:3,0,"TR,SC
210 REM --LEGGE BYTE CONCATENAMENTO--
215 PRINT#1,"B-P:3,0"
220 CB=CB+1:GET#3,A$,B$:GOSUB380
225 REM --STAMPA UN PALLINO PER OGNI BLOCCO--
230 TR=ASC(A$):SC=ASC(B$):PRINT"●";
235 IFTR<>0THEN205
240 REM --CONTROLLO LUNGHEZZA FILE--
245 IFCB=NBTHEN325
250 REM --CICLO DI ATTESA--
255 IFCB=NBTHENFORI=1TO250:NEXTI:GOTO85
260 PRINT:PRINTM2$FI$
265 REM --SE CONTEGGIO ERRATO--
270 PRINT"□DEVO AZZERARE  ? (□S■/□N■)"
275 GETA$:IFA$=""THEN275
280 IFA$="N"THEN400
285 IFA$<>"S"THEN275
290 PRINT"□AZZERAMENTO E VALIDATE"
295 PRINT#1,"S:"FI$:PRINT#1,"V"
300 CLOSE3:CLOSE2:CLOSE1:RUN
305 REM --PASSA A PROSSIMA ENTRATA--
310 FI=FI+1:IFFI=8THENGOSUB350:FI=0
315 GOTO175
320 REM --TUTTO BENE--
325 PRINT:PRINTTAB(9)"□□□□OK":FI=FI+1
330 IFFI=8THENGOSUB350:FI=0
335 GOTO255
```

```
340 REM --RE LETTURA DIRECTORY--
345 REM --TD=0 A FINE DIRECTORY--
350 IFTD=0THENPRINTM3$:GOTO400
355 REM --LETTURA BLOCCO--
360 A$="":B$="":PRINT#1,"U1:2,0";TD;SD
365 GET#2,A$,B$:GOSUB380
370 TD=ASC(A$):SD=ASC(B$):RETURN
375 REM --SISTEMAZIONE BYTE LETTI--
380 IFB$=""THENB$=CHR$(0)
385 IFA$=""THENA$=CHR$(0)
390 RETURN
395 REM --CHIUSURA FILE--
400 CLOSE3:CLOSE2:CLOSE1:STOP
```

Il programma legge la directory e i file in modo diretto, usando gli indirizzi di traccia e settore. Abbiamo abbondato in REM, quindi non aggiungiamo commento. Ti suggeriamo di estendere il programma per fargli controllare anche i file di altro tipo, tenendo conto che:

- per poter controllare i file RANDOM, è necessario che essi siano registrati come USR nel modo da noi suggerito;
- per i file RELATIVI il controllo è semplice per quanto riguarda i blocchi del file principale, ma si deve andare a prelevare dall'entrata della directory anche l'indirizzo del primo blocco side sector e procedere con il controllo anche per questa parte.

Il programma TRAC/SET , che segue, è ricavato dal DISPLAY T&S del TEST-/DEMO, apportando alcune modifiche. Con esso si possono stampare gruppi di settori concatenati sui primi due byte o settori singoli.

```
5 REM TRAC/SET
10 RC$="      ■VIDEO ■■■■■■■■■■D  ■■■P■RINTER"
15 RD$="      ■■VIDEO ■"
20 RE$="      ■■PRINTER■"
25 PRINT"■■■──────────────────────────────"
30 PRINT"  CONTENUTO BLOCCHI   "
35 PRINT"──────────────────────────────"
40 REM COSTANTI
45 SP$=" ":NL$=CHR$(0)
50 HX$="0123456789ABCDEF"
```

```
55 FS$="":FORI=64 TO 95
60 FS$=FS$+"▐"+CHR$(I)+"▌":NEXT I
65 SS$="  ":FOR I=192 TO 223
70 SS$=SS$+"▐"+CHR$(I)+"▌":NEXT I
75 DIM A$(15),NB(2)
80 D$="0"
85 PRINTRC$
90 GETJJ$:IF JJ$="" THEN90
95 IF JJ$="V"THENPRINTRD$
100 IF JJ$="P"THENPRINTRE$:OPEN4,4
110 GOSUB450
115 REM CARICA BUFFER
120 INPUT"▐▐▐TRACCIA, SETTORE";T,S
125 IF T=0 OR T>35 THEN355
130 IF JJ$="V" THENGOSUB360:GOTO135
131 PRINT#4:PRINT#4,"TRACCIA"T" SETTORE"S:PRINT#4
135 PRINT#15,"U1:2,"D$;T;S:GOSUB335
140 PRINT#15,"B-P:2,0"
145 REM LEGGE BYTE 0
150 GET#2,A$(0):IFA$(0)=""THENA$(0)=NL$
155 PRINT#15,"B-P:2,1"
160 IF JJ$="V"THEN170
165 IF JJ$="P"THEN230
170 REM VIDEO
175 K=1:NB(1)=ASC(A$(0))
180 FORJ=0TO63:IFJ=32THENGOSUB375:GOTO365
185 FOR I=K TO 3
190 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
195 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
200 NEXT I:K=0
205 A$="":B$=":":N=J*4:GOSUB 405:A$=A$+":"
210 FOR I=0 TO 3:N=ASC(A$(I)):GOSUB 405
215 C$=A$(I):GOSUB 425:B$=B$+C$
220 NEXT I:IF JJ$="V" THEN PRINTA$B$
225 NEXT J:GOTO295
230 REM PRINTER
235 K=1:NB(1)=ASC(A$(0))
240 FOR J=0 TO 15
245 FOR I=K TO 15
250 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
255 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
260 NEXT I:K=0
```

```
265 A$="":B$=":":N=J*16:GOSUB 405:A$=A$+":"
270 FOR I=0 TO 15:N=ASC(A$(I)):GOSUB 405
275 C$=A$(I):GOSUB 425:B$=B$+C$
280 NEXT I
285 IF JJ$="P" THEN PRINT#4,A$B$
290 NEXT J:GOTO295
295 REM SUCCESSIVO BLOCCO
300 PRINT"TRACCIA/SETT.SEGUENTE"NB(1)NB(2) "▯"
305 PRINT"DESIDERI PROSEGUIRE        S/N   "
310 GET Z$:IF Z$="" THEN310
315 IF Z$<>"S" THEN325
320 T=NB(1):S=NB(2):GOSUB445:GOSUB450:GOTO125
325 IF Z$="N" THEN GOSUB445:GOSUB450:GOTO120
330 GOTO 310
335 REM ROUTINE ERRORE
340 INPUT#15,EN,EM$,ET,ES:IF EN=0 THEN RETURN
345 PRINT"▯ERRORE DISCO▯"EN,EM$,ET,ES
350 END
355 PRINT#15,"I"D$:GOSUB445:CLOSE4:PRINT"END":END
360 PRINT"▯▯▯TRACCIA"T" SETTORE"S"▯":RETURN
365 IF Z$="N"THEN J=80:GOTO 225
370 GOTO185
375 REM MESS. CONTINUAZIONE
380 PRINT"▯▯▯▯▯CONTINUO(S/N)"
385 GETZ$:IF Z$="" THEN 385
390 IF Z$="N" THEN RETURN
395 IF Z$<>"S" THEN 385
400 PRINT"▯TRACCIA" T " SETTORE"S:RETURN
405 REM CONV.HEX
410 A1=INT(N/16):A$=A$+MID$(HX$,A1+1,1)
415 A2=INT(N-16*A1):A$=A$+MID$(HX$,A2+1,1)
420 A$=A$+SP$:RETURN
425 REMCONV.ASCII
430 IF ASC(C$)<32 THEN C$=" ":RETURN
435 IF ASC(C$)<128 OR ASC(C$)>159 THEN RETURN
440 C$=MID$(SS$,3*(ASC(C$)-127),3):RETURN
445 CLOSE2:CLOSE15:RETURN
450 OPEN15,8,15,"I"+D$:GOSUB335
455 OPEN2,8,2,"#":GOSUB335:RETURN
```

Il programma SIST/DISCO è una modifica sostanziale di CHECK DISK; esso lascia il dischetto in grado di essere usato e con allocati, solo nella BAM, i settori guasti. Su un dischetto in queste condizioni non può essere eseguita la funzione VALIDATE.

```
5 REM SIST/DISCO
10 REM TABELLA TRACCE E SETTORI GUASTI
15 DIMT(100):DIMS(100)
20 DIMG1(21),G2(19),G3(18),G4(17)
25 DATA0,10,20,8,18,6,16,4,14,2,12,9,19
30 DATA7,17,5,15,3,13,1,11
35 DATA0,11,1,12,2,13,3,14,4,15,5,16,6,17
40 DATA7,18,8,9,10
45 DATA0,10,1,11,2,12,3,13,4,14,5,15
50 DATA6,16,7,17,8,9
55 DATA0,10,1,11,2,12,3,13,4,14,5
60 DATA15,6,16,7,8,9
65 FORJ=1TO21:READG1(J):NEXTJ
70 FORJ=1TO19:READG2(J):NEXTJ
75 FORJ=1TO18:READG3(J):NEXTJ
80 FORJ=1TO17:READG4(J):NEXTJ
85 RC$=" BLOCCHI GUASTI SONO STATI ALLOCATI"
90 RD$="BLOCCHI GUASTI":RE$="TRACCIA"
95 RF$="SETTORE"
100 PRINT"──────────────────────────"
105 PRINT"      SIST/DISCO      "
110 PRINT"──────────────────────────"
115 OPEN15,8,15
120 PRINT#15,"V0"
125 N%=RND(TI)*255
130 A$="":FORI=1TO255
135 A$=A$+CHR$(255AND(I+N%)):NEXT
140 OPEN2,8,2,"#":GOSUB325
145 PRINT:PRINT#2,A$;
150 J=1
155 FORT=1TO17:FORL=1TO21
160 S=G1(L):GOSUB200:NEXTL:NEXTT
165 FORT=18TO24:FORL=1TO19
170 S=G2(L):GOSUB200:NEXTL:NEXTT
175 FORT=25TO30:FORL=1TO18
180 S=G3(L):GOSUB200:NEXTL:NEXTT
```

```
185 FORT=31TO35:FORL=1TO17
190 S=G4(L):GOSUB200:NEXTL:NEXTT
195 GOTO260
200 PRINT#15,"B-A:0"T;S
205 INPUT#15,EN,EM$,ET,ES
210 IFEN=0THEN225
215 IFEN=65THENRETURN
220 STOP
225 PRINT#15,"U2:2,0"T;S
230 NB=NB+1
235 INPUT#15,EN,EM$,ES,ET
240 IF EN=0THENRETURN
245 T(J)=T:S(J)=S:J=J+1
250 PRINT"■■■BLOCCHI GUASTI:";T;S
255 RETURN
260 PRINT#15,"V0"
265 GOSUB325
270 CLOSE2
275 IFJ=1THEN350
280 OPEN2,8,2,"#"
285 PRINTRD$,RE$,RF$
295 FORI=1TOJ-1
300 PRINT#15,"B-A:0"T(I);S(I)
305 PRINT,,T(I),S(I)
310 NEXT
315 PRINT"■";J-1;RC$
320 CLOSE2:CLOSE15:END
325 INPUT#15,EN,EM$,ET,ES
330 IF EN=0 THEN RETURN
335 PRINT"■■■ERRORE#"EN,EM$;ET;ES"■"
340 PRINT#15,"I0"
345 RETURN
350 PRINT"■■■■■■TUTTO BENE!    ":CLOSE15:END
```

La ragione che rende molto veloce questo programma sta nel modo ottimizzato di allocare e scrivere i diversi blocchi. Abbiamo preparato con le frasi DATA e i quattro vettori G1, G2, G3 e G4 una traccia ottimizzata per l'allocazione dei settori.

Segue il programma NUMBUFFER; esso serve per vedere quanti buffer possono essere aperti contemporaneamente in modo diretto e quali sono i loro numeri (da 0 a 7).

```
5 REM NUMBUFFER
10 OPEN15,8,15,"I"
15 INPUT"QUANTI BUFFER";N
20 A$="":B$="":C$="":D$="":E$=""
25 ONNGOTO50,45,40,35,30
30 OPEN2,8,2,"#":GOSUB115:GET#2,E$:GOSUB115
35 OPEN3,8,3,"#":GOSUB115:GET#3,D$:GOSUB115
40 OPEN4,8,4,"#":GOSUB115:GET#4,C$:GOSUB115
45 OPEN5,8,5,"#":GOSUB115:GET#5,B$:GOSUB115
50 OPEN6,8,6,"#":GOSUB115:GET#6,A$:GOSUB115
55 ONNGOTO100,90,80,70,60
60 IFE$=""THENE$=CHR$(0)
65 PRINTASC(E$):CLOSE2
70 IFD$=""THEND$=CHR$(0)
75 PRINTASC(D$):CLOSE3
80 IFC$=""THENC$=CHR$(0)
85 PRINTASC(C$):CLOSE4
90 IFB$=""THENB$=CHR$(0)
95 PRINTASC(B$):CLOSE5
100 IFA$=""THENA$=CHR$(0)
105 PRINTASC(A$):CLOSE6
110 CLOSE15:STOP
115 INPUT#15,EN,EM$,ET,ES
120 PRINTEN;EM$;ET;ES
125 RETURN
```

Come puoi vedere, provando il programma, se cerchi di aprire più di 4 canali, hai la segnalazione NO CHANNEL. L'istruzione GET#lfn, usata subito dopo la OPEN..."#", fornisce il numero del buffer assegnato. Se provi il programma dopo averlo caricato con LOAD trovi che, per 4 canali, vengono assegnati i buffer: 2, 1, 0, 4. Se, invece, dopo il LOAD del programma, togli corrente all'unità 1541, ottenendo il reset della RAM, poi ridai corrente e RUN, ottiene, sempre per 4 canali, i numeri di buffer 3, 2, 1, 0.

Segue il programma INDBUFFER; esso serve per vedere quali indirizzi della RAM dell'unità 1541 sono assegnati ai diversi buffer.

```
5 REM INDBUFFER
10 DIMR$(8,7):FORK=1TO8:FORI=1TO7
15 R$(K,I)="":NEXTI:NEXTK
16 M1$="IND. IN MEMORIA DEL BUFFER: "
20 OPEN7,4:PRINT#7,"RISULTATI INDBUFFER"
25 OPEN15,8,15,"I"
30 INPUT"QUANTI BUFFER";N:PRINT#7
35 PRINT#7,N;" BUFFER":PRINT#7
40 A$="":B$="":C$="":D$="":E$=""
45 ONNGOTO70,65,60,55,50
50 OPEN2,8,2,"#":GOSUB270:GET#2,E$:GOSUB270
55 OPEN3,8,3,"#":GOSUB270:GET#3,D$:GOSUB270
60 OPEN4,8,4,"#":GOSUB270:GET#4,C$:GOSUB270
65 OPEN5,8,5,"#":GOSUB270:GET#5,B$:GOSUB270
70 OPEN6,8,6,"#":GOSUB270:GET#6,A$:GOSUB270
75 PRINT#7,"NUMERI DEI BUFFER: ";
80 ONNGOTO165,145,125,105,85
85 IFE$=""THENE$=CHR$(0)
90 PRINT#7,ASC(E$);
95 PRINT#15,"B-P:"2;1
100 PRINT#2,"22222";:GOSUB270
105 IFD$=""THEND$=CHR$(0)
110 PRINT#7,ASC(D$);
115 PRINT#15,"B-P:"3;1
120 PRINT#3,"33333";:GOSUB270
125 IFC$=""THENC$=CHR$(0)
130 PRINT#7,ASC(C$);
135 PRINT#15,"B-P:"4;1
140 PRINT#4,"44444";:GOSUB270
145 IFB$=""THENB$=CHR$(0)
150 PRINT#7,ASC(B$);
155 PRINT#15,"B-P:"5;1
160 PRINT#5,"55555";:GOSUB270
165 IFA$=""THENA$=CHR$(0)
170 PRINT#7,ASC(A$);
175 PRINT#15,"B-P:"6;1
180 PRINT#6,"66666";:GOSUB270
185 PRINT#7
190 FORK=1TO8:M=K-1
195 FORI=1TO7:L=I-1
200 PRINT#15,"M-R"CHR$(L)CHR$(M)
205 GET#15,R$(K,I)
```

```
210 NEXTI
215 NEXTK
220 FORK=1TO8:FORI=1TO7
225 IFR$(K,I)=""THENR$(K,I)=CHR$(0)
230 NEXTI:NEXTK
235 FORK=1TO8
240 PRINT#7,M1$;STR$((K-1)*256)
245 PRINT#7,"PRIMI 8 BYTE DEL BUFFER:";
250 FORI=1TO7
255 PRINT#7,R$(K,I);:NEXTI:PRINT#7
260 NEXTK
265 FORK=1TO7:CLOSEK:NEXTK:CLOSE15:STOP
270 INPUT#15,EN,EM$,ET,ES
275 PRINTEN;EM$;ET;ES
280 RETURN
```

Il programma fa le seguenti cose:

- chiede quanti buffer si vogliono, apre il numero richiesto di file (massimo 5) e, per ognuno, legge con GET# il numero del buffer e lo memorizza;
- scrive per ogni file aperto 5 caratteri di riconoscimento nel buffer a partire dalla posizione 1;
- legge con M-R i primi 7 caratteri di ogni buffer (da 0 a 7), usando gli indirizzi iniziali: 0, 256, 512, 768, 1024, 1280, 1536, 1792 (i primi 4K di RAM) e li memorizza nella matrice R$(8,7);
- stampa i risultati.

Riportiamo i risultati ottenuti con due prove; la prima si riferisce a un RUN senza reset dell'unità 1541, la seconda a un RUN dopo il reset.

```
RISULTATI INDBUFFER

 4  BUFFER

NUMERI DEI BUFFER:  2  1  0  4
IND. IN MEMORIA DEL BUFFER:  0
PRIMI 8 BYTE DEL BUFFER:
IND. IN MEMORIA DEL BUFFER:  256
PRIMI 8 BYTE DEL BUFFER:A
IND. IN MEMORIA DEL BUFFER:  512
PRIMI 8 BYTE DEL BUFFER:M-R
```

```
IND. IN MEMORIA DEL BUFFER:  768
PRIMI 8 BYTE DEL BUFFER:55555)
IND. IN MEMORIA DEL BUFFER:  1024
PRIMI 8 BYTE DEL BUFFER:44444N
IND. IN MEMORIA DEL BUFFER:  1280
PRIMI 8 BYTE DEL BUFFER:33333I
IND. IN MEMORIA DEL BUFFER:  1536
PRIMI 8 BYTE DEL BUFFER:
IND. IN MEMORIA DEL BUFFER:  1792
PRIMI 8 BYTE DEL BUFFER:
```

```
RISULTATI INDBUFFER

 4  BUFFER

NUMERI DEI BUFFER:  3  2  1  0
IND. IN MEMORIA DEL BUFFER:  0
PRIMI 8 BYTE DEL BUFFER:
IND. IN MEMORIA DEL BUFFER:  256
PRIMI 8 BYTE DEL BUFFER:A
IND. IN MEMORIA DEL BUFFER:  512
PRIMI 8 BYTE DEL BUFFER:M-R

IND. IN MEMORIA DEL BUFFER:  768
PRIMI 8 BYTE DEL BUFFER:66666
IND. IN MEMORIA DEL BUFFER:  1024
PRIMI 8 BYTE DEL BUFFER:55555
IND. IN MEMORIA DEL BUFFER:  1280
PRIMI 8 BYTE DEL BUFFER:44444
IND. IN MEMORIA DEL BUFFER:  1536
PRIMI 8 BYTE DEL BUFFER:33333
IND. IN MEMORIA DEL BUFFER:  1792
PRIMI 8 BYTE DEL BUFFER:
```

APPENDICE A

MAGIC DESK I

Questo package, realizzato in modo piuttosto divertente, consente di usare i file su disco anche senza saper scrivere programmi che li gestiscono.

Il package risiede su cartridge; dopo l'inserimento, a macchina spenta, quando dai corrente compare sul video un menù quadro. Vengono proposte le diverse funzioni sotto forma di disegni in piacevoli colori. Per usare il programma è necessario inserire un Joystick nella porta 2. I comandi si scelgono usando il Joystick per muovere nelle 4 direzioni la mano che compare sul video. Per scegliere una funzione si deve toccare con il dito della mano, mobile sul video, l'oggetto che la rappresenta; premendo poi il bottone del fuoco la funzione va in esecuzione.
Il sistema necessario per usare il package in tutte le sue possibilità è composto da: il calcolatore, l'unità disco, la stampante, il video e un joystick.

Per sapere cosa si può fare, basta premere il tasto CBM; compare sul video un menù in parte disegnato e in parte descritto. Devi inserire un dischetto nuovo, la prima volta che usi il programma, nell'unità 1541.

Le operazioni possibili sono:

- Scrivere a macchina usando la tastiera del calcolatore e vedendo sul videc quello che si scrive. Il calcolatore si trasforma in una vera macchina da scrivere, con attivabili tutte le normali funzioni di tale apparecchiatura.

- Predisporre l'orologio, che avanza in tempo reale, all'ora che si desidera.

- Memorizzare i documenti scritti (funzione di archiviazione) in uno dei 3 cassetti raccoglitori visibili, e in una determinata cartelletta.

• Richiamare sul video documenti già archiviati per correggerli, stamparli, e, se si vuole, conservarli di nuovo.

• Se un documento scritto non interessa più, si può buttarlo nel cestino della carta straccia.

Segue un listato di una lettera scritta e stampata con il programma.

Spett.
Lettore del libro
COMMODORE 64 - I FILE

Milano, giugno 1984

Caro Lettore,

spero che questo manuale ti sia utile, e che tu abbia imparato ad usare bene i file sia su cassetta che su disco.

Comunque, puoi sempre servirti di MAGIC DESK I per risolvere i tuoi problemi.

Gradisci i miei migliori saluti

Rita Bonelli

APPENDICE B

CALC RESULT

È un ottima versione del FOGLIO ELETTRONICO. Viene fornito un manuale esauriente, una ROM da inserire nel calcolatore e un dischetto.

L'utente ha a disposizione sul video una matrice formata da 64 colonne (denominate: a,b,c,...,aa,ab,ac,...,ba,bb,...,bk) e da 254 righe (denominate: 1,2,...,254). In realtà sul video è visibile solo una parte di questo grande foglio elettronico, ma, servendosi dei tasti di movimento del cursore, si può far comparire la parte che interessa della matrice.
In ogni casella della matrice si può scrivere:

- una LABEL (descrizione),
- un numero,
- una formula di calcolo.

Ogni casella può essere facilmente modificata; si può definire la larghezza in caratteri di ogni colonna della matrice.
Le formule di calcolo lavorano sulle altre caselle della matrice (che sono le variabili). Sono disponibili anche alcune funzioni definite dal sistema; tra esse MIN, MAX, IF...THEN...ELSE, funzioni matematiche e logiche.
Se si modifica un numero, contenuto in una casella coinvolta in una formula, automaticamente viene ricalcolato il risultato.
Il contenuto di un foglio elettronico può esssere salvato su disco e richiamato quando serve; esso può essere stampato tutto o in parte.
Sono gestibili contemporaneamente più fogli elettronici, fino a 32. È possibile costruire un foglio prendendo dati da altri fogli.
Il video può essere diviso in due parti (finestre) gestibili separatamente.

Riportiamo un semplicissimo esempio di bilancio familiare su 3 mesi.

	GENN.	FEBBR.	MARZO
AFFITTO	300000	300000	300000
VITTO	600000	650000	700000
LUCE	70000	80000	90000
GAS	20000	15000	18000
TELEF.	80000	90000	100000
AUTO	100000	120000	150000
SCUOLE	60000	60000	60000
TOTALE	1230000	1315000	1418000

Il programma si presta a svariate applicazioni come:

- bilanci,
- piani di ammortamento,
- ripartizione di spese,
- gestione del conto corrente,
- previsioni finanziarie,
- stampa di grafici.

Si possono creare dei modelli di matrice per diverse applicazioni, memorizzarli su disco con nomi opportuni e servirsene per lavorare.

Inoltre è possibile generare dei file di dati di tipo DIF (Data Interchange Format), che possono essere utilizzati da altri programmi scritti in BASIC.

APPENDICE C

LE BASI DI DATI

Sono disponibili sul mercato diversi package per la gestione sofisticata di archivi (basi di dati). Essi fanno le poche, ma fondamentali, cose, che noi abbiamo trattato nei capitoli precedenti, più tante altre. I programmi componenti il pacchetto sono scritti in ASSEMBLER, quindi la loro velocità di esecuzione non è paragonabile con quella dei nostri programmi in BASIC.

Vogliamo qui citare due package, che abbiamo avuto modo di provare; essi sono:

SUPERBASE 64 della Precision Software,
MAGPIE DATABASE, distribuito dalla JCE nella versione italiana.

Si tratta di due prodotti pregevoli; essi non solo consentono di generare e gestire archivi di dati con uno o più indici di ricerca, ma consentono anche di svolgere calcoli e di richiamare programmi che operano sui record degli archivi.

L'utente può predisporre i tracciati dei record dei file, i tracciati dei quadri video e i formati dei prospetti da stampare.

I manuali che accompagnano i programmi sono di notevole mole, e ci vuole un pò di tempo per impadronirsi di tutte le possibilità dei package. Si tratta comunque di prodotti che possono essere usati anche da persone inesperte di programmazione, ma non digiune dei concetti fondamentali inerenti il trattamento dei dati con il calcolatore.

6

INDICE

Capitolo 1 - LA GRAFICA
1.1 Capacità grafiche del Commodore 64 1
1.2 Le operazioni logiche AND e OR 1
1.3 Locazioni grafiche 4
1.3.1 Selezione del banco 4
1.3.2 Memoria del video 5
1.3.3 Memoria del colore 6
1.3.4 Memoria dei caratteri 7
1.3.5 Memoria della pagina grafica 8
1.3.6 Sovrapposizione di un'area di memoria grafica alla ROM 9

Capitolo 2 - CARATTERI
2.1 Caratteri definibili 11
2.1.1 Copia dei caratteri della ROM 12
2.1.2 Programmi per la creazione dei caratteri 18
2.1.3 Caratteri a sfondo programmabile 30
2.1.4 Caratteri multicolore 40

Capitolo 3 - PAGINA GRAFICA
3.1 Pagina grafica 51
3.1.1 Pagina grafica ad alta risoluzione 51
3.1.2 Pagina grafica multicolore 72

Capitolo 4 - ALTRE POSSIBILITA'
4.1 Altre possibilità del VIC II 75
4.1.1 Annullamento dello schermo (Screen Blanking) 75
4.1.2 Il registro di linea e i registri di interrupt 76
4.1.3 Scorrimento fine (Smooth Scrolling) 80

Capitolo 5 - UN PICCOLO SISTEMA GRAFICO
5.1 La pagina grafica 91
5.2 L'area dei parametri 92
5.3 Ingresso nel modo grafico ad alta risoluzione 93
5.4 PLOT/UNPLOT/INVERT 97
5.5 Linee orizzontali e verticali 102
5.6 Idee 105

CAPITOLO 1

LA GRAFICA

1.1 CAPACITA' GRAFICHE DEL COMMODORE 64

La grafica è una tra le più affascinanti applicazioni nel mondo dei calcolatori. Il tuo COMMODORE 64 possiede delle capacità grafiche talmente potenti da eguagliare quelle di calcolatori molto più costosi. Hai a disposizione:

- un video composto da ben 64000 puntini (pixel) disposti su 200 righe di 320 punti;
- la possibilità di sovrapporre all'immagine del video delle figure mobili, comodissime nelle animazioni grafiche;
- 16 bellissimi colori che ti aiutano a creare grafici di sicuro effetto.

In questo capitolo vedremo come definire caratteri diversi da quelli disponibili all'accensione e come gestire i 64000 punti che abbiamo a disposizione.
Nel Capitolo 3 vedremo come si possono usare le figure mobili (SPRITE).

1.2 LE OPERAZIONI LOGICHE AND E OR

Per poter parlare di grafica con il calcolatore è assolutamente necessario conoscere queste due operazioni logiche che sono essenziali anche in altre applicazioni del calcolatore.
Cominciamo col definire le operazioni AND e OR tra due bit:

0 AND 0 = 0	0 OR 0 = 0
0 AND 1 = 0	0 OR 1 = 1
1 AND 0 = 0	1 OR 0 = 1
1 AND 1 = 1	1 OR 1 = 1

Il risultato della AND tra due bit è quindi uguale a 1 solo se il primo E il secondo bit sono a 1; il risultato della OR tra due bit è uguale a 1 se il primo O il secondo bit (o entrambi) sono uguali a 1.
Ogni bit del risultato di una AND (o di una OR) tra bytes è dato dalla AND (o la OR) tra i bit corrispondenti degli operandi, ad esempio:

3 AND 2 = 2

infatti 3 in binario si esprime come 11 mentre 2 come 10 quindi, partendo dal bit meno significativo:

1 AND 0 = 0 e 1 AND 1 = 1

risultato 10, cioè 2 in decimale.

Ma come mai sono così importanti queste strane operazioni nella grafica col calcolatore? Perchè esse permettono di MASCHERARE alcuni bit nello scrivere o nel leggere un byte di memoria. Ad esempio, vogliamo sapere il contenuto del bit di posizione 4 (cioè il quinto da destra verso sinistra) di un certo byte? Bene, non dobbiamo far altro che fare l'operazione AND del byte con 16 (2 elevato a 4); se il risultato è uguale a 0 allora il bit è a zero, se è 16 (cioè 2 elevato a 4) allora il bit è a 1; infatti:

7 6 5 4 3 2 1 0 posizioni bit nel byte

? ? ? 0 ? ? ? ? (byte che vogliamo mascherare)
0 0 0 1 0 0 0 0 (16 in binario)
0 0 0 0 0 0 0 0 (0, risultato della AND)

? ? ? 1 ? ? ? ? (byte che vogliamo mascherare)
0 0 0 1 0 0 0 0 (16 in binario)
0 0 0 1 0 0 0 0 (16, risultato della AND)

Altro esempio: vogliamo porre a 1 il bit di posizione 3 di un byte, lasciando inalterati gli altri bit? Questa volta facciamo la OR tra il byte e 8 (2 elevato a 3); in questo modo il bit di posizione 3 del risultato diventa 1 (sia 0 che 1 OR 1 danno 1) e gli altri rimangono inalterati (1 OR 0 dà 1, 0 OR 0 dà 0).
Perdonaci ma, data l'importanza dell'argomento, vogliamo fare un ultimo esempio: dobbiamo porre a 0 il bit di posizione 2 di un certo byte. Come facciamo? Dobbiamo fare la AND tra questo byte e il numero 11111011 (255-4=251), infatti, in questo modo, il bit di posizione 2 verrà posto a 0 (sia 0 che 1 AND 0 danno 0) e gli altri bit rimarranno inalterati (0 AND 1 dà 0 ,1 AND 1 dà 1). Pensiamo ora un pò al nostro COMMODORE 64 e scriviamo il programma GRAF1.

```
1 REM GRAF1
10 N=4:M$="ESATTO !!!"
20 A=RND(0)
30 A=INT(RND(1)*2↑N)
40 B=INT(RND(1)*2↑N)
50 C=INT(RND(0)*2)
60 PRINT"QUANTO FA'"A;
70 D=A:GOSUB1000
80 PRINTTAB(20)B$
90 IFC=0THENPRINT"AND";:GOTO110
100 PRINT"OR";
110 PRINTB;
120 D=B:GOSUB1000
130 PRINTTAB(20)B$" ";
140 INPUTR
150 IFR=(AANDB)ANDC=0THENPRINTM$:PRINT:GOTO30
160 IFR=(AORB)ANDC=1THENPRINTM$:PRINT:GOTO30
170 PRINT"NO, IL RISULTATO E' ";
180 IFC=0THENR=AANDB:GOTO200
190 R=AORB
200 D=R:GOSUB1000
210 PRINTB$" ="R:PRINT:GOTO30
1000 B$=""
1010 FORI=N-1TO0STEP-1
1020 IF(DAND2↑I)=0THENB$=B$+"0":GOTO1040
1030 B$=B$+"1"
1040 NEXT
1050 RETURN
```

A cosa serve lo avrai già immaginato: ad allenarti con le operazioni AND e OR! Intanto vediamo come funziona, poi dai RUN al tuo calcolatore e buon divertimento!

COMMENTO A GRAF1.
Linea 10: pone uguale a 4 il numero di bit degli operandi: ti consigliamo di cominciare ad allenarti con 4 bit per poi passare ad operazioni a 8 bit (di più non servono per i nostri scopi); crea la costante M$.
Linea 20: sorteggia la base per i numeri random.
Linea 30: sceglie un numero intero tra 0 e 2 elevato a (N-1) come primo operando.

Linea 40: sceglie un numero intero tra 0 e 2 elevato a (N-1) come secondo operando.
Linea 50: sceglie fra 0 e 1; se 0 l'operazione proposta è AND, altrimenti l'operazione proposta è OR.
Linee 60-130: formula la domanda scrivendo gli operandi in decimale e in binario, usando la routine in 1000 che converte un numero decimale posto nella variabile D, nel corrispondente numero binario e lo pone nella string B$.
Linea 140: accetta la risposta.
Linee 150-160: controlla la risposta; se esatta lo segnala e riparte.
Linee 170-210: calcola e scrive la risposta esatta in decimale e binario e riparte.
Linea 1000: annulla la stringa che conterrà il numero binario.
Linee 1010-1040: calcola, partendo dal bit più significativo, il valore di ciascun bit facendo la AND tra il numero decimale e una maschera composta da tutti zeri tranne il bit che interessa, il risultato viene posto in B$.
Linea 1050: torna al programma principale.

1.3 LOCAZIONI GRAFICHE

Una delle caratteristiche del COMMODORE 64 che lo rende molto versatile nella programmazione grafica e ne permette una buona gestione della memoria, è il fatto che le locazioni grafiche (mappa video, mappa dei caratteri ecc.) non sono vincolate sempre alla stessa zona di memoria; infatti è possibile cambiarle scrivendo l'indirizzo che interessa negli appositi registri del processore video 6566/6567 (VIC II). Nel seguito abbiamo raccolto le spiegazioni delle operazioni necessarie a cambiare le locazioni grafiche. Se non hai già una certa conoscenza dei modi grafici del COMMODORE 64, ti consigliamo di continuare la lettura dal Capitolo 2 e tornare a questi paragrafi quando trovi un richiamo.

1.3.1 Selezione del banco

Il bus indirizzi del VIC II è di soli 14 bit (può contenere al massimo il numero 16383), questo vuol dire che il VIC II può indirizzare solo 16 Kbyte di memoria alla volta. E' possibile selezionare quale dei 4 banchi di 16 Kbyte, presenti nel COMMODORE 64, debba essere VISTO dal VIC II. Il numero di banco è scritto, negato, nei due bit meno significativi del registro 0 (indirizzo 56576 (DD00H)) del secondo integrato di I/O, 6526 (CIA). Supponiamo, per esempio, che in quei due bit sia scritto il numero 01 (binario): negandolo otteniamo il numero 10, in questo caso quindi, il banco selezionato è il terzo; il numero B che distingue i banchi varia da 0 a 3. L'istruzione che serve per cambiare il banco è questa:

POKE 56576,(PEEK(56576)AND252)+N

Per trovare il valore desiderato di N consulta la Tabella 1.1.

BANCO	INDIRIZZI decimali	INDIRIZZI esadecimali	N
0	0-16383	0000H-3FFFH	3
1	16384-32767	4000H-7FFFH	2
2	32768-49151	8000H-BFFFH	1
3	49152-65535	C000H-FFFFH	0

Tabella 1.1 Valore di N per la selezione del banco

1.3.2 Memoria del video

L'indirizzo del primo byte della memoria video è dato da:

B*16384+V*1024

dove B è il banco selezionato (vedi Paragrafo 1.3.1) e V è il contenuto dei 4 bit più significativi del registro 24 del VIC II (indirizzo 53272 (D018H)). Il comando che serve per cambiare la locazione della memoria video è questo:

POKE53272,PEEK((53272)AND15)+16*N

Per trovare il valore desiderato di N consulta la Tabella 1.2.

LOCAZIONI DEL VIDEO decimali	LOCAZIONI DEL VIDEO esadecimali	N
0- 1023	0000H-03FFH	0
1024- 2047	0400H-07FFH	1
2048- 3071	0800H-0BFFH	2

3072- 4095	0C00H-0FFFH	3
4096- 5119	1000H-13FFH	4
5120- 6143	1400H-17FFH	5
6144- 7167	1800H-1BFFH	6
7168- 8191	1C00H-1FFFH	7
8192- 9215	2000H-23FFH	8
9216-10239	2400H-27FFH	9
10240-11263	2800H-2BFFH	10
11264-12287	2C00H-2FFFH	11
12288-13311	3000H-33FFH	12
13312-14335	3400H-37FFH	13
14336-15359	3800H-3BFFH	14
15360-16383	3C00H-3FFFH	15

Tabella 1.2 Valore di N per il posizionamento della memoria video

Ricorda che agli indirizzi scritti in tabella devi aggiungere il numero di banco moltiplicato per 16384 (4000H) (Paragrafo 1.3.1).
Nota che se cambi la locazione della mamoria video devi avvisare il SISTEMA OPERATIVO perchè sappia dove scrivere. Per far questo devi memorizzare V/256, nel byte 648 (288H), dove V è l'indirizzo del primo byte della memoria video.

1.3.3 Memoria del colore

La memoria del colore è formata da 1000 byte di memoria con solo 4 bit attivi (bastano infatti 4 bit per scegliere un colore tra 16 possibili); essa si trova dall'indirizzo 55296 (D800H) all'indirizzo 56295 (DBE7H) e non può essere spostata. E' da notare che questa zona di memoria fa parte dei 20 Kbyte DOPPI nel COMMODORE 64. Per accedere alla RAM che risponde agli indirizzi della memoria del colore bisogna azzerare i 2 bit meno significativi del registro di I/O della CPU (indirizzo 1) disabilitando contemporaneamente le ROM del BASIC e del SISTEMA OPERATIVO. Questa operazione, che può esser fatta, ovviamente, solo da un programma in linguaggio macchina, esclude anche l'I/O, rendendo accessibile la RAM da 53248 a 57343 (D000H - DFFFH).

1.3.4 Memoria dei caratteri

L'indirizzo del primo byte della memoria dei caratteri è dato da:
B*16384+C*2048
dove B è il banco selezionato (Paragrafo 1.3.1) e C è il contenuto dei bit 3, 2 e 1 del registro 24 del VIC II (indirizzo 53272 (D018H)). Questa regola non è più valida in questi quattro casi:

1) B=0 e C=2
2) B=0 e C=3
3) B=2 e C=2
4) B=2 e C=3

In questi casi, infatti la memoria dei caratteri è la ROM generatrice. Poichè il bit di posizione 0 del registro 24 del VIC II non ha alcuna funzione, non lo devi mascherare quando cambi la locazione della memoria dei caratteri. Il comando che devi dare al tuo COMMODORE 64 sarà quindi:

POKE53272,PEEK((53272)AND240)+N

Per trovare il valore desiderato di N consulta la Tabella 1.3, nella quale, per maggior chiarezza, abbiamo scritto anche il numero B del banco.

LOCAZIONI DEI CARATTERI decimali	esadecimali	N	BANCO
0- 2047	0000H-07FFH	0	0
2048- 4095	0800H-0FFFH	2	0
ROM GENERATRICE		4	0
ROM GENERATRICE		6	0
8192-10239	2000H-27FFH	8	0
10240-12287	2800H-2FFFH	10	0
12288-14335	3000H-37FFH	12	0
14336-16383	3800H-3FFFH	14	0
16384-18431	4000H-47FFH	0	1
18432-20479	4800H-4FFFH	2	1
20480-22527	5000H-57FFH	4	1
22528-24575	5800H-5FFFH	6	1

24576-26623	6000H-67FFH	8	1
26624-28671	6800H-6FFFH	10	1
28672-30719	7000H-77FFH	12	1
30720-32767	7800H-7FFFH	14	1
32768-34815	8000H-87FFH	0	2
34816-36863	8800H-8FFFH	2	2
ROM GENERATRICE		4	2
ROM GENERATRICE		6	2
40960-43007	A000H-A7FFH	8	2
43008-45055	A800H-AFFFH	10	2
45056-47103	B000H-B7FFH	12	2
47104-49151	B800H-BFFFH	14	2
49152-51199	C000H-C7FFH	0	3
51200-53247	C800H-CFFFH	2	3
53248-55295	D000H-D7FFH	4	3
55296-57343	D800H-DFFFH	6	3
57344-59391	E000H-E7FFH	8	3
59392-61439	E800H-EFFFH	10	3
61440-63487	F000H-F7FFH	12	3
63488-65535	F800H-FFFFH	14	3

Tabella 1.3 Valore di N per il posizionamento della memoria dei caratteri

Nota che il valore del bit di posizione 1 del registro 24 del VIC II viene cambiato ogni volta che premi i tasti COMMODORE e SHIFT insieme. Prova a eseguire:

```
PRINT PEEK(53272)
```

con il set di caratteri maiuscoli e poi:

```
print peek(53272)
```

con il set di caratteri minuscoli.
Vedrai che la risposta nel secondo caso è maggiore di due. Se questo fatto potesse provocarti dei problemi, nei tuoi programmi puoi inserire l'istruzione:

```
PRINT CHR$(8)
```

che non permetterà più di cambiare il set di caratteri da tastiera; per riabilitare SHIFT + CBM dai il comando PRINT CHR$ (9).

1.3.5 Memoria della pagina grafica

La memoria della pagina grafica è formata da 8000 byte di memoria. L'indirizzo della prima di queste locazioni è dato da B*16384+P*8192, dove B è il banco selezionato (vedi paragrafo 1.3.1) e P è il contenuto del bit 3 del registro 24 del VIC

II (indirizzo 53272 (D018H)). La pagina grafica, essendo così estesa, può essere messa solo in due posizioni diverse per ogni banco: i primi 8000 byte o i primi 8000 byte della seconda metà.
Ovviamente, nel banco 0 non è possibile porre la pagina grafica nella prima posizione, perchè verrebbe a sovrapporsi alla pagina 0; inoltre nella prima posizione del banco 2 e del banco 0 il VIC VEDE tra gli indirizzi 38912 e 43007 (9800H-A7FFH) la ROM generatrice dei caratteri (vedi paragrafo precedente). Per cambiare indirizzo della pagina grafica le istruzioni sono:

POKE 53272,PEEK(53272)AND247 per la prima posizione nel banco,

POKE 53272,PEEK(53272)OR8 per la seconda posizione nel banco.

Nota che per cambiare posizione della pagina grafica si usa un bit che interessa anche la posizione della memoria dei caratteri: quando torni al modo caratteri devi quindi ricordarti di porre in quel bit il valore corretto.

1.3.6 Sovrapposizione di un'area di memoria grafica alla ROM

Come avrai notato il COMMODORE 64 ti dà la possibilità di porre la memoria video, la memoria dei caratteri e la memoria della pagina grafica in locazioni in cui rispondono le ROM del BASIC e del SISTEMA OPERATIVO. Cosa succede in tal caso? Il VIC II, che può solamente leggere la memoria, legge la RAM che risponde a quegli indirizzi e non sicuramente la ROM (non può interessare al VIC II il contenuto della ROM del BASIC o del SISTEMA OPERATIVO); alla CPU, invece, interessano i contenuti di queste ROM, per cui, quando accede a quegli indirizzi per leggere, legge la ROM, ma quando scrive sa di non riuscire a scrivere su una ROM, quindi scrive sulla RAM. Come vedi, anche se allochi un'area di memoria grafica in indirizzi dedicati alle ROM, tutto va bene perchè la CPU riesce a scrivere e il VIC II riesce a leggere. Usare un'area di questo tipo comporta il vantaggio che non si RUBA memoria al programma e alle variabili BASIC, ma non si riesce a sapere il contenuto dei byte di RAM poichè la CPU non riesce a leggerli. Si può risolvere questo piccolo problema con una routine in linguaggio macchina che disabiliti la ROM interessata (deve anche trascurare gli interrupt) e permetta alla CPU di leggere la RAM.

CAPITOLO 2

CARATTERI

2.1 CARATTERI DEFINIBILI

Tu sai già come il COMMODORE 64 possa visualizzare tutti i caratteri che puoi vedere sul video, e sai anche che le IMMAGINI di questi caratteri (una serie di 8 byte per ogni carattere) sono gelosamente custodite in ROM (memoria a sola lettura), perchè non vengano perse ogni volta che spegni il calcolatore. Può sembrare, quindi che ogni volta che verrà premuto il tasto "A" del COMMODORE 64 il video mostrerà una A. Invece il COMMODORE 64 ti dà anche la possibilità di creare uno per uno tutti i caratteri. Proviamo a definire un carattere: per esempio il simbolo CBM che appare sul tasto in basso a sinistra della tastiera del tuo calcolatore. Usando la stessa tecnica, con cui sono disegnati normalmente i caratteri, costruiamo una matrice 8 per 8 e riempiamola in modo da ottenere il simbolo voluto, usando 0 per i punti spenti e 1 per quelli accesi:

0	0	0	1	1	0	0	0
0	0	1	1	1	0	0	0
0	1	1	0	0	1	1	1
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1
0	0	1	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

Ora ci tocca fare un pò di conti: la prima riga (cioè il primo byte del carattere) contiene il numero binario 00011000, cioè 24 decimale. Facendo lo stesso calcolo per le altre righe otteniamo una serie di 8 numeri cioè 24, 56, 103, 102, 103, 56, 24, 0.
Ci basta ora cambiare al VIC II l'indirizzo della mappa dei caratteri (Paragrafi 1.3.1 e 1.3.4) e memorizzare gli 8 numeri ottenuti nei primi 8 byte della nuova mappa dei caratteri, programmando il primo carattere, quello che corrisponde al D-CODE 0, e che all'accensione rappresenta la chiocciola.
Nota che, creando il carattere, abbiamo cercato di non usare mai uno 0 o un 1 ISOLATO (cioè con un simbolo opposto sia a destra che a sinistra): infatti se usi un televisore o un monitor di qualità non ottima, è facile che, non facendo così, appaiano sullo schermo effetti non desiderati. Dopo aver capito bene il meccanismo di programmazione dei caratteri, prova a creare un carattere contenente in tutti i byte il numero 85 ,cioè 01010101 binario, e guarda se crea strani effetti sul tuo monitor.
Proviamo ora il programma GRAF2.

```
1 REM GRAF2
10 POKE53280,0:POKE53281,0
15 POKE53272,(PEEK(53272)AND240)+12
20 FORI=0TO7:READA:POKE12*1024+I,A:NEXT
30 PRINTCHR$(154)CHR$(142)CHR$(147)"@"
40 GETA$:IFA$=""THEN40
50 POKE53272,(PEEK(53272)AND240)+5
60 DATA24,56,103,102,103,56,24,0
```

Fai girare il programma: vedrai sullo schermo, in alto a sinistra il carattere che abbiamo appena programmato. Se premi un tasto tutto torna normale e vedrai che il carattere torna ad essere quello che è normalmente: una chiocciola. Se invece di premere un tasto qualunque premi il tasto di STOP e provi a scrivere qualche cosa, lo schermo si riempirà di caratteri incomprensibili (non ti preoccupare, premendo STOP-RESTORE tutto tornerà alla normalità); questi caratteri sono tutti i caratteri che non abbiamo programmato, infatti ci siamo preoccupati di riempire solo i primi 8 byte della nuova mappa.

COMMENTO A GRAF2.
Linea 10: pone schermo e bordo neri e scrive 12 nel nibble (4 bit) meno significativo del registro 24 del VIC II, senza cambiare il valore degli altri 4 bit (sposta la mappa dei caratteri a 12288 (3000H)).
Linea 20: riempe la matrice del primo carattere (la chiocciola) con i dati che abbiamo calcolato prima.
Linea 30: pulisce lo schermo e scrive in blù chiaro, nel set maiuscolo, una chiocciola. (CHR$(8) disabilita la funzione dei tasti COMMODORE-SHIFT).
Linea 40: attende che sia premuto un tasto.
Linea 50: scrive 5 nel nibble meno significativo del registro 24 del VIC II, senza cambiare il valore degli altri 4 bit (sposta nuovamente la mappa dei caratteri agli indirizzi della ROM).
Linea 60: contiene gli 8 dati che formano il carattere.

2.1.1 Copia dei caratteri dalla ROM

Non sempre, quando definisci un nuovo set di caratteri, vuoi ottenere un risultato completamente diverso dal contenuto della ROM: potrebbe capitarti di volere avere a disposizione tutte le lettere del set del COMMODORE 64 e cambiarne solo i simboli grafici, o tenere i caratteri POSITIVI (normali) e cambiare quelli in campo inverso. Cosa fare in questo caso? Ovviamente dovrai copiare su RAM la parte del

contenuto della ROM che ti interessa e, quindi, definire come abbiamo già visto gli altri caratteri. L'unico problema è dato dal fatto che la CPU non VEDE, normalmente, la ROM generatrice di caratteri. Per far si che la CPU possa leggere da questa ROM bisogna porre a zero il bit di posizione 2 del registro di I/O della CPU stessa (chè risponde all'indirizzo 1), eseguendo il comando:
POKE 1, PEEK(1) AND 251.
Purtroppo questa operazione disabilita l' I/O cioè i due CIA (gli integrati di I/O), il SID (l'integrato del suono) e il VIC II poichè la ROM generatrice risponde proprio agli indirizzi (53248-57343 (D000-DFFF)) cui rispondono i registri di questi integrati. Non è quindi conveniente tenere sempre inserita la ROM a scapito di tutti gli integrati di I/O. Poichè, inoltre, la routine di interrupt ha bisogno di accedere alle locazioni di I/O, per poter trasferire il contenuto della ROM dei caratteri in RAM, dovremo anche disabilitare l'interrupt eseguendo l'istruzione:
POKE 56334, PEEK (56334) AND 254
cioè ponendo a zero il bit di posizione 0 del registro 14 del CIA 1 che ferma il contatore che genera l'interrupt. Prova ora il programma GRAF3.

```
1 REM GRAF3
5 PRINTCHR$(142)
10 POKE56,48:POKE55,0:CLR
20 A=PEEK(56334):POKE56334,AAND254
30 POKE1,PEEK(1)AND251
40 FORI=0TO2047:POKEI+12288,PEEK(I+53248):NEXT
50 POKE1,PEEK(1)OR4
60 POKE56334,A
70 POKE53272,(PEEK(53272)AND240)+12
```

Il programma occuperà il tuo COMMODORE 64 per circa 35 secondi, e quando avrà finito dirà semplicemente "READY". Nulla sembrerà essere successo, ma in realtà ora i dati relativi ai caratteri non vengono più dalla ROM ma dalla RAM. Prova infatti a scrivere una chiocciola sullo schermo e poi esegui il comando:
FOR I=0 TO 7:POKE 12288+I,0:NEXT
appena premi RETURN la chiocciola sparirà dallo schermo e non sarà più possibile scriverne un'altra poichè hai appena cancellato i dati relativi alla chiocciola in RAM. Prova poi a selezionare il set delle minuscole e vedrai che quei caratteri non sono stati programmati.

COMMENTO A GRAF3.
Linea 5: seleziona il set delle maiuscole.
Linea 10: pone la fine della memoria riservata al BASIC all'indirizzo 12288 (3000H)

perchè il programma e le variabili non sporchino i caratteri programmati che partiranno, appunto, dall'indirizzo 12288.
Linea 20: pone il contenuto del registro 14 del CIA 1 nella variabile A e disabilita l'interrupt.
Linea 30: esclude l'I/O per abilitare la ROM dei caratteri.
Linea 40: trasferisce metà dei caratteri della ROM (il set delle maiuscole: dall'indirizzo 53248 all'indirizzo 55295) in RAM a partire dall'indirizzo 12288.
Linea 50: esclude la ROM riabilitando l'I/O.
Linea 60: pone nel registro 14 del CIA 1 il valore iniziale per riabilitare l'interrupt.
Linea 70: pone l'inizio della mappa dei caratteri in 12288.

Prova ora a scrivere il programma GRAF4 il cui compito è proprio quello di creare un nuovo set di caratteri che sia uguale a quello in ROM per i caratteri normali, e diverso per quelli in campo inverso.

```
1 REM GRAF4
5 PRINTCHR$(142)
10 POKE56,48:POKE55,0:CLR:S=111713:NC=128
17 M$=" ECCO I NUOVI CARATTERI DEL COMMODORE !"
20 A=PEEK(56334):POKE56334,AAND254
30 POKE1,PEEK(1)AND251
40 FORI=0TO1023:POKEI+12288,PEEK(I+53248):NEXT
50 POKE1,PEEK(1)OR4
60 POKE56334,A
70 FORI=1TONC*8
80 READA:B=B+A
90 POKEI+13311,A
100 NEXTI
105 IFB<>STHENPRINT"ERRORE NELLE LINEE DATA":STOP
110 PRINTCHR$(147)
120 POKE53272,(PEEK(53272)AND240)+12
130 PRINTCHR$(17)CHR$(18)M$
1000 DATA120,254,238,238,232,224,254,126
1010 DATA56,124,238,254,254,238,238,238
1020 DATA252,238,238,252,238,238,254,252
1030 DATA124,254,230,224,224,238,254,124
1040 DATA248,252,238,238,238,238,252,248
1050 DATA254,254,224,248,248,224,254,254
1060 DATA254,254,224,248,248,224,224,224
1070 DATA62,126,224,238,238,230,126,62
```

```
1080 DATA238,238,238,254,254,238,238,238
1090 DATA254,254,56,56,56,56,254,254
1100 DATA254,254,56,56,56,248,248,48
1110 DATA238,236,252,248,248,236,238,238
1120 DATA224,224,224,224,224,224,254,254
1130 DATA198,238,254,254,254,214,198,198
1140 DATA198,230,246,254,254,222,206,198
1150 DATA124,254,238,238,238,238,254,124
1160 DATA252,254,230,254,252,224,224,224
1170 DATA124,238,238,238,238,254,124,14
1180 DATA252,230,230,254,248,252,238,230
1190 DATA124,254,224,252,126,14,254,124
1200 DATA254,254,56,56,56,56,56,56
1210 DATA238,238,238,238,238,238,254,124
1220 DATA238,238,238,238,238,254,124,56
1230 DATA198,198,214,254,254,254,238,198
1240 DATA238,238,124,56,56,124,238,238
1250 DATA238,238,238,124,56,56,56,56
1260 DATA254,254,30,60,120,240,254,254
1270 DATA124,120,112,112,112,112,120,124
1280 DATA15,31,48,255,255,112,255,255
1290 DATA62,30,14,14,14,14,30,62
1300 DATA0,24,60,126,24,24,24,24
1310 DATA0,16,48,127,127,48,16,0
1320 DATA0,0,0,0,48,48,0,0
1330 DATA60,60,60,60,60,0,60,60
1340 DATA238,238,238,0,0,0,0,0
1350 DATA102,255,255,102,255,255,102,0
1360 DATA24,254,224,255,7,127,124,24
1370 DATA227,231,238,28,56,119,231,199
1380 DATA60,102,62,120,239,230,127,61
1390 DATA14,28,56,96,0,0,0,0
1400 DATA60,120,240,240,240,240,120,60
1410 DATA60,30,15,15,15,15,30,60
1420 DATA24,90,60,255,60,90,24,0
1430 DATA0,56,56,254,254,56,56,0
1440 DATA0,0,0,0,0,24,56,112
1450 DATA0,0,0,126,126,0,0,0
1460 DATA0,0,0,0,0,56,56,56
1470 DATA3,7,14,28,56,112,224,192
1480 DATA124,254,238,238,238,254,254,124
1490 DATA56,56,120,120,56,56,254,254
```

```
1500 DATA124,254,134,12,56,96,254,254
1510 DATA124,254,14,60,60,14,254,124
1520 DATA28,60,124,204,254,254,12,12
1530 DATA254,224,252,14,14,206,254,124
1540 DATA124,238,224,252,238,238,254,124
1550 DATA254,238,28,56,56,56,56,56
1560 DATA124,238,238,124,238,238,254,124
1570 DATA124,238,238,126,14,206,254,124
1580 DATA0,56,56,0,0,56,56,0
1590 DATA0,0,56,56,0,56,56,112
1600 DATA2,14,62,254,254,62,14,2
1610 DATA0,0,126,126,0,126,126,0
1620 DATA64,112,124,127,127,124,112,64
1630 DATA60,126,102,14,28,28,0,28
1640 DATA0,0,0,255,255,0,0,0
1650 DATA8,28,62,127,127,28,62,0
1660 DATA24,24,24,24,24,24,24,24
1670 DATA0,0,0,255,255,0,0,0
1680 DATA0,0,255,255,0,0,0,0
1690 DATA0,255,255,0,0,0,0,0
1700 DATA0,0,0,0,255,255,0,0
1710 DATA48,48,48,48,48,48,48,48
1720 DATA12,12,12,12,12,12,12,12
1730 DATA0,0,0,224,240,56,24,24
1740 DATA24,24,28,15,7,0,0,0
1750 DATA24,24,56,240,224,0,0,0
1760 DATA192,192,192,192,192,192,255,255
1770 DATA192,224,112,56,28,14,7,3
1780 DATA3,7,14,28,56,112,224,192
1790 DATA255,255,192,192,192,192,192,192
1800 DATA255,255,3,3,3,3,3,3
1810 DATA0,60,126,126,126,126,60,0
1820 DATA0,0,0,0,0,255,255,0
1830 DATA54,127,127,127,62,28,8,0
1840 DATA96,96,96,96,96,96,96,96
1850 DATA0,0,0,7,15,28,24,24
1860 DATA195,231,126,60,60,126,231,195
1870 DATA0,60,126,102,102,126,60,0
1880 DATA24,24,102,102,24,24,60,0
1890 DATA6,6,6,6,6,6,6,6
1900 DATA8,28,62,127,62,28,8,0
1910 DATA24,24,24,255,255,24,24,24
```

```
1920 DATA192,192,48,48,192,192,48,48
1930 DATA24,24,24,24,24,24,24,24
1940 DATA0,0,3,62,118,54,54,0
1950 DATA255,127,63,31,15,7,3,1
1960 DATA0,0,0,0,0,0,0,0
1970 DATA240,240,240,240,240,240,240,240
1980 DATA0,0,0,0,255,255,255,255
1990 DATA255,255,0,0,0,0,0,0
2000 DATA0,0,0,0,0,0,0,255
2010 DATA192,192,192,192,192,192,192,192
2020 DATA204,204,51,51,204,204,51,51
2030 DATA3,3,3,3,3,3,3,3
2040 DATA0,0,0,0,204,204,51,51
2050 DATA255,254,252,248,240,224,192,128
2060 DATA3,3,3,3,3,3,3,3
2070 DATA24,24,24,31,31,24,24,24
2080 DATA0,0,0,0,15,15,15,15
2090 DATA24,24,24,31,31,0,0,0
2100 DATA0,0,0,248,248,24,24,24
2110 DATA0,0,0,0,0,0,255,255
2120 DATA0,0,0,31,31,24,24,24
2130 DATA24,24,24,255,255,0,0,0
2140 DATA0,0,0,255,255,24,24,24
2150 DATA24,24,24,248,248,24,24,24
2160 DATA192,192,192,192,192,192,192,192
2170 DATA224,224,224,224,224,224,224,224
2180 DATA7,7,7,7,7,7,7,7
2190 DATA255,255,0,0,0,0,0,0
2200 DATA255,255,255,0,0,0,0,0
2210 DATA0,0,0,0,0,255,255,255
2220 DATA3,3,3,3,3,3,255,255
2230 DATA0,0,0,0,240,240,240,240
2240 DATA15,15,15,15,0,0,0,0
2250 DATA24,24,24,248,248,0,0,0
2260 DATA240,240,240,240,0,0,0,0
2270 DATA0,0,255,255,0,0,0,8
```

COMMENTO A GRAF4.

Le linee dalla 5 alla 60 sono uguali a quelle del programma GRAF3 che abbiamo appena visto, tranne che nei seguenti particolari: nella linea 10 inizializziamo le variabili S e NC che servono per fare un controllo sulle linee DATA (S=somma di

tutti i valori contenuti nei data, NC=numero di caratteri che si vogliono programmare); nella linea 40 il loop finisce a 1023 invece che a 2047 perchè non copiamo i caratteri in campo inverso.
Linea 70: inizializza un loop di NC*8 cicli (di lunghezza diversa in dipendenza dal numero di caratteri che si vogliono programmare).
Linea 80: legge uno dopo l'altro i dati delle linee DATA e ne fa la somma ponendola nella variabile B.
Linea 90: mette i dati letti nei byte da 13312 in poi (parte che sarà dedicata ai caratteri in campo inverso).
Linea 100: chiude il loop.
Linea 105: controlla che la somma sia uguale alla variabile S.
Linea 110: pulisce lo schermo.
Linea 120: pone l'inizio della mappa dei caratteri in 12288.
Linea 130: scrive, una linea più in basso, in campo inverso (cioè con il nuovo set di caratteri).
Linee 1000-2270: 128 linee di 8 dati ciascuna: ogni linea contiene i dati relativi ad ognuno dei 128 caratteri in campo inverso.

Copiare questo programma, con 128 linee DATA, può essere piuttosto monotono: abbiamo quindi calcolato il valore che S deve avere se si programmano solo 32, 64 o 96 caratteri. Per NC=32, S vale 47284; per NC=64, S vale 72982 e per NC=96, S vale 90667. Puoi così programmare solo 32 caratteri e, se il risultato è di tuo gradimento, programmare gli altri in un secondo tempo, o a più riprese.

2.1.2 Programmi per la creazione dei caratteri

Programmare nuovi caratteri con il COMMODORE 64 è facile ma non proprio molto immediato: è difficile cioè VEDERE che 24, 56, 103, 102, 103, 56, 24, 0 rappresentano un simbolo grafico. Sarebbe molto più comodo se si potesse disegnare direttamente sullo schermo il carattere e farlo memorizzare senza dover fare tanti calcoli, PEEK e POKE. Il programma GRAF5 fà precisamente questo. E' commentato come al solito, e puoi quindi eliminare certe parti che non ti interessano e magari aggiungerne altre per renderlo più adatto ai tuoi scopi.

```
1 REM GRAF5
10 POKE13*4096+32,0:POKE13*4096+33,0
15 PRINTCHR$(154)CHR$(142)CHR$(8)
20 POKE56,48:POKE55,0:CLR
```

```
25 US$="            PREMI F1 PER USCIRE"
26 US$=CHR$(147)+US$+CHR$(31)
100 PRINTCHR$(147)
110 PRINT"OPZIONI :":PRINT:PRINT
120 PRINT"1 CREA CARATTERE":PRINT
130 PRINT"2 MEMORIZZA CARATTERE":PRINT
140 PRINT"3 CORREGGE CARATTERE":PRINT
150 PRINT"4 MOSTRA CARATTERI":PRINT
160 PRINT"5 SALVA SET DI CARATTERI":PRINT
170 PRINT"6 CARICA SET DI CARATTERI":PRINT
180 PRINT"7 FINE"
190 GETA$:IFVAL(A$)=0THEN190
195 I=VAL(A$)
200 ONIGOSUB1000,2000,3000,4000,5000,6000,7000
210 GOTO100
1000 PRINTUS$
1010 FORI=1TO8:PRINTCHR$(17);:NEXTI
1020 FORI=1TO8
1030 FORJ=1TO16:PRINTCHR$(32);:NEXT
1040 FORJ=1TO8:PRINTCHR$(43);:NEXTJ
1050 PRINT:NEXTI
1060 PC=55672:XC=0:YC=0
1070 POKEPC,1
1080 GETA$:IFA$=""THEN1080
1090 A=ASC(A$):SP=0
1100 IFA=29ANDXC<7THENXC=XC+1:PC=PC+1:SP=1
1110 IFA=157ANDXC>0THENXC=XC-1:PC=PC-1:SP=-1
1120 IFA=17ANDYC<7THENYC=YC+1:PC=PC+40:SP=40
1130 IFA=145ANDYC>0THENYC=YC-1:PC=PC-40:SP=-40
1140 IFA=32THENPOKEPC-54272,160
1150 IFA=20THENPOKEPC-54272,43
1160 IFA=133THENPRINTCHR$(154):POKEPC-SP,6
1165 IFA=133THEN1180
1170 POKEPC-SP,6:GOTO1070
1180 FORI=0TO7
1190 BY(I)=0
1200 FORJ=0TO7
1210 BY(I)=BY(I)-2↑J*(PEEK(1400+7-J+40*I)=160)
1220 NEXTJ:NEXTI
1230 RETURN
2000 PRINTCHR$(147)
2010 PRINT"1 NORMALE":PRINT
```

```
2020 PRINT"2 REVERSATO":PRINT
2030 PRINT"3 SIMMETRIA VERTICALE":PRINT
2040 PRINT"4 SIMMETRIA ORIZZONTALE":PRINT
2050 PRINT"5 RUOTATO DI 90 GRADI ";
2055 PRINT"IN SENSO ORARIO":PRINT
2060 GETA$:IFA$=""THEN2060
2070 A=VAL(A$):IFA=0ORA>5THEN2060
2080 INPUT"CARATTERE NUMERO ";NC%
2090 IFNC%>511ORN%<0THEN2080
2100 ONAGOSUB2200,2300,2400,2500,2600
2110 FORI=0TO7:POKE12288+8*NC%+I,BY(I):NEXT
2120 RETURN
2200 RETURN
2300 FORI=0TO7:BY(I)=255-BY(I):NEXT
2310 RETURN
2400 FORI=0TO7
2410 FORJ=0TO3
2420 BD=(BY(I)AND2↑J)/2↑J:BY(I)=BY(I)-2↑J*BD
2430 BS=(BY(I)AND2↑(7-J))/2↑(7-J)
2435 BY(I)=BY(I)-2↑(7-J)*BS
2440 BY(I)=BY(I)+2↑J*BS:BY(I)=BY(I)+2↑(7-J)*BD
2450 NEXTJ,I
2460 RETURN
2500 FORI=0TO3
2510 T=BY(I)
2520 BY(I)=BY(7-I)
2530 BY(7-I)=T
2540 NEXTI
2550 RETURN
2600 FORI=0TO7
2610 B2(I)=BY(I):BY(I)=0
2620 NEXTI
2630 FORI=0TO7
2640 FORJ=0TO7
2645 EX=(B2(I)AND2↑(7-J))/2↑(7-J)
2650 BY(J)=BY(J)OR(2↑(EX*I)*-(EX<>0))
2660 NEXTJ,I
2670 RETURN
3000 PRINTCHR$(147)
3010 INPUT"CARATTERE DA CORREGGERE ";NC%
3020 IFNC%>511ORN%<0THEN3010
3030 PRINTUS$
```

```
3040 FORI=1TO8:PRINTCHR$(17);:NEXTI
3050 FORI=0TO7
3060 FORJ=1TO16:PRINTCHR$(32);:NEXT
3070 FORJ=0TO7
3080 IFPEEK(12288+8*NC%+I)AND2↑(7-J)THENSI=-1
3085 IFSITHENPRINTCHR$(18)CHR$(32)CHR$(146);
3086 IFSITHENSI=0:GOTO3100
3090 PRINTCHR$(43);
3100 NEXTJ:PRINT:NEXTI
3110 GOTO1060
4000 PRINTCHR$(147)"PREMI F1 PER TORNARE"
4005 PRINTCHR$(17)"PREMI I TASTI CORRISPONDENTI"
4006 PRINT"AI CARATTERI CHE VUOI VEDERE"
4010 FORI=1TO2000:NEXTI
4020 PRINTCHR$(147);
4030 POKE53272,(PEEK(53272)AND240)+12
4040 GETA$:IFA$=""THEN4040
4045 IFA$=CHR$(133)THENF1=-1
4050 IFF1THENPOKE53272,(PEEK(53272)AND240)+5
4055 IFF1THENF1=0:PRINTCHR$(154):RETURN
4060 PRINTA$;
4070 GOTO4040
5000 PRINTCHR$(147)
5010 PRINT"DISCO O NASTRO ?"
5015 GETDV$:IFDV$<>"N"ANDDV$<>"D"THEN5015
5020 PRINT
5030 INPUT"SET NUMERO ";NS%
5040 PRINT
5050 PRINT"FINO A CHE CARATTERE VUOI SALVARE ";
5055 INPUTNC%
5060 IFNC%>511THEN5040
5070 IFDV$="N"THEN5500
5080 OPEN1,8,2,"@:SET #"+STR$(NS%)+",S,W"
5082 GOSUB9000
5085 IFNNTHENFORI=1TO2000:NEXT:CLOSE1:CLOSE15
5087 IFNNTHENRETURN
5090 PRINT#1,CHR$(NC%);:FORI=0TO(NC%+1)*8-1
5100 PRINT#1,CHR$(PEEK(12288+I));
5110 NEXTI
5120 CLOSE1:CLOSE15
5130 RETURN
```

```
5500 OPEN1,1,1,"SET #"+STR$(NS%)
5510 PRINT#1,CHR$(NC%);:FORI=0TO(NC%+1)*8-1
5520 PRINT#1,CHR$(PEEK(12288+I));
5530 NEXTI
5540 CLOSE1
5550 RETURN
6000 PRINTCHR$(147)
6010 PRINT"DISCO O NASTRO ?"
6015 GETDV$:IFDV$<>"N"ANDDV$<>"D"THEN6015
6020 PRINT
6030 INPUT"SET NUMERO ";NS%
6070 IFDV$="N"THEN6500
6080 OPEN1,8,2,"SET #"+STR$(NS%)+",S,R"
6082 GOSUB9000
6085 IFNNTHENFORI=1TO2000:NEXT:CLOSE1:CLOSE15
6087 IFNNTHENRETURN
6090 GET#1,A$:NC%=ASC(A$):FORI=0TO(NC%+1)*8-1
6100 GET#1,A$:POKE12288+I,ASC(A$+CHR$(0))
6110 NEXTI
6120 CLOSE1:CLOSE15
6130 RETURN
6500 OPEN1,1,0,"SET #"+STR$(NS%)
6510 FORI=0TO(NC+1)%*8-1
6520 GET#1,A$:POKE12288+I,ASC(A$+CHR$(0))
6530 NEXTI
6540 CLOSE1
6550 RETURN
7000 PRINTCHR$(17)"SICURO ?"
7010 GETA$:IFA$=""THEN7010
7020 IFA$="S"THENSYS58260
7030 RETURN
9000 PRINT:OPEN15,8,15:INPUT#15,NN,MM$,TT,SS
9005 PRINTNN,MM$,TT,SS
9010 RETURN
```

Il programma GRAF5 può essere diviso in 15 sezioni ben distinte:

- 1: le linee 10-26 inizializzano il calcolatore.
- 2: le linee dalla 100 alla 210 mostrano il menù e saltano alla funzione richiesta.
- 3: le linee dalla 1000 alla 1230 ti permettono di disegnare un nuovo carattere e riempiono un vettore con i dati relativi al carattere disegnato (i calcoli li fa il programma).

- 4: le linee dalla 2000 alla 2120 mostrano il menù di memorizzazione dei caratteri, saltano alla routine per il tipo di operazione richiesta sul vettore e memorizzano il carattere.
- 5: la linea 2200 lascia il vettore come è; è stata messa per rispondere alla chiamata della linea 2100.
- 6: le linee dalla 2300 alla 2310 pongono nel vettore il NEGATIVO del carattere già contenuto.
- 7: le linee dalla 2400 alla 2460 pongono nel vettore il carattere che gode di simmetria verticale rispetto a quello già contenuto.
- 8: le linee dalla 2500 alla 2550 pongono nel vettore il carattere che gode di simmetria orizzontale rispetto a quello già contenuto.
- 9: le linee dalla 2600 alla 2670 pongono nel vettore il carattere già contenuto, ruotato di 90 gradi in senso orario. Chiamando tre volte questa routine otterrai nel vettore il carattere ruotato di 90 gradi in senso antiorario.
- 10: le linee dalla 3000 alla 3110 riempono il vettore con i dati relativi al carattere che vuoi correggere e, saltando alla linea 1060, ti permettono di correggere questo carattere.
- 11: le linee dalla 4000 alla 4070 ti permettono di vedere i caratteri che hai creato.
- 12: le linee dalla 5000 alla 5550 salvano su disco o su nastro i caratteri che hai programmato, assegnando al file un nome di riconoscimento.
- 13: le linee dalla 6000 alla 6550 caricano da disco o da nastro un set di caratteri salvato precedentemente.
- 14: le linee dalla 7000 alla 7030 chiudono il programma.
- 15: le linee dalla 9000 alla 9010 scrivono lo stato del disco.

COMMENTO A GRAF5 SEZIONE PER SEZIONE.

SEZIONE 1.

Linee 10-15: pone schermo e sfondo neri, il colore del cursore blu chiaro, seleziona il set delle maiuscole e disabilita la funzione dei tasti CBM e SHIFT.
Linea 20: pone i puntatori di fine memoria a 12288 (3000H) per non sporcare con le variabili i caratteri che verranno creati.
Linee 25-26: inizializza la costante US$.

SEZIONE 2.

Linee 100-180: mostra le opzioni.
Linea 190: accetta un carattere numerico compreso tra 1 e 7.
Linea 195: pone nella variabile I il valore corrispondente al tasto premuto.
Linea 200: salta alla routine richiesta.
Linea 210: torna a mostrare le opzioni.

SEZIONE 3.
E' la sezione più complicata del programma: per comprenderne bene il funzionamento è preferibile suddividerla in 3 sottosezioni:
A) linee 1000-1050: disegna una griglia 8 per 8 su cui disegnare il carattere.
B) linee 1060-1170 e 1240: gestisce la posizione del cursore in seno alla griglia.
C) linee 1180-1230: riempie il vettore BY(7) con i dati relativi al carattere disegnato sulla griglia.

Torniamo al commento linea per linea.
Linea 1000: pulisce lo schermo, scrive in alto PREMI F1 PER USCIRE e pone il cursore di colore blu.
Linea 1010: sposta il cursore più in basso di 8 linee.
Linea 1020: per 8 volte esegue fino alla linea 1050.
Linea 1030: sposta il cursore a destra di 16 posizioni.
Linea 1040: scrive 8 volte +.
Linea 1050: porta il cusore a nuova linea e chiude il loop.
Linea 1060:pone l'indirizzo del byte della RAM del colore, corrispondente al + in alto a sinistra, nella variabile PC (posizione cursore), pone a zero le variabili XC e YC (coordinate X e Y del cursore nella griglia).
Linea 1070: la posizione del cursore diventa bianca (colore 1).
Linea 1080: accetta dalla tastiera un carattere qualunque.
Linea 1090: A = codice ASCII del carattere ricevuto da tastiera e SP = 0 (SP=spostamento).
Linea 1100: se il carattere ricevuto da tastiera è CURSORE A DESTRA e il cursore non è all'estrema destra della griglia (XC=7) allora incrementa XC e PC e pone SP = 1.
Linea 1110: se il carattere ricevuto da tastiera è CURSORE A SINISTRA e il cursore non è all'estrema sinistra della griglia (XC=0) allora decrementa XC e PC e pone SP = -1.
Linea 1120: se il carattere ricevuto da tastiera è CURSORE IN BASSO e il cursore non è nell'ultima linea della griglia (YC=7) allora incrementa YC, somma 40 a PC e pone SP = 40 (40 è il numero di caratteri contenuti in una linea dello schermo).
Linea 1130: se il carattere ricevuto da tastiera è CURSORE IN ALTO e il cursore non è nella prima linea della griglia (YC=0) allora decrementa YC, sottrae 40 da PC e pone SP = -40.
Linea 1140: se il carattere ricevuto da tastiera è SPAZIO pone nella posizione della mappa video corrispondente alla posizione del cursore uno spazio in campo inverso in modo da ANNERIRE un punto della griglia (il numero 54272 è la differenza di indirizzi tra mappa del colore e mappa video).
Linea 1150: se il carattere ricevuto da tastiera è DELETE pone nella posizione della mappa video corrispondente alla posizione del cursore un + in modo da CANCELLARE un punto della griglia.
Linea 1160: se il carattere ricevuto da tastiera è F1 rimette il cursore al colore blu

chiaro, colora di blu la posizione del cursore della griglia (colore 6).
Linea 1165: se il carattere ricevuto è F1 salta alla sottosezione C.
Linea 1170: ogni altro carattere viene ignorato; il programma colora la vecchia posizione del cursore di blu (se lo spostamento è stato nullo colora la posizione corrente del colore) e salta alla linea 1070 dove colora la posizione corrente del cursore di bianco.
Linea 1180: per ogni riga della griglia esegue fino a 1220.
Linea 1190: pone a zero il corrispondente elemento del vettore.
Linea 1200: per ogni elemento di una riga esegue fino a 1220.
Linea 1210: somma alla variabile corrispodente alla riga interessata 2 elevato alla posizione dell'elemento di riga considerato (partendo da destra); se in tale posizione vi è uno spazio in campo inverso, altrimenti 0 (nota che se esegui l'istruzione PRINT A=B il risultato è -1 se A=B, 0 se A < > B).
Linea 1220: chiude il ciclo degli elementi e quello delle righe. A questo punto nel vettore BY sono contenuti gli 8 numeri necessari per programmare il carattere disegnato sulla griglia.
Linea 1230: fine della routine.

SEZIONE 4.
Linee 2000-2055: mostra le opzioni relative alla memorizzazione dei caratteri.
Linee 2060-2070: accetta dalla tastiera un numero tra 1 e 5 e lo pone nella variabile A.
Linee 2080-2090: accetta un numero tra 0 e 511 e lo pone nella variabile NC% (il carattere che si vuole programmare).
Linea 2100: salta alla routine di modifica del vettore richiesta.
Linea 2110: memorizza il carattere ponendo in 8 byte di memoria a partire da 12288 + NC% * 8 il contenuto del vettore.
Linea 2120: torna dalla routine.

SEZIONE 5.
Linea 2200: è stata messa per rispondere alla chiamata della linea 2100.

SEZIONE 6.
Linea 2300: pone in ogni elemento del vettore (255-il valore già contenuto); questa operazione, compiuta su numeri minori di 256 (come nel nostro caso), nega gli 8 bit del numero.
Linea 2310: torna dalla routine.

SEZIONE 7.
Linea 2400: per ogni elemento del vettore esegue fino a 2450.
Linea 2410: per J da 0 a 3 esegue fino a 2450.
Linea 2420: pone il valore del bit j-esimo dell'elemento considerato del vettore nella variabile BD (bit di destra) e lo azzera.

Linea 2430-2435: pone il valore del bit (7-j)-esimo dell'elemento considerato del vettore nella variabile BS (bit di sinistra) e lo azzera.
Linea 2440: pone il bit di sinistra al posto del bit di destra e viceversa.
Linea 2450: chiude i loop di J e degli elementi del vettore.
Linea 2460: torna dalla routine.

SEZIONE 8.
Linea 2500: per i primi 4 elementi del vettore esegue fino a 2540.
Linea 2510: pone nella variabile T (temporanea) il valore dell'elemento considerato.
Linea 2520: pone nell'elemento considerato il valore dell'elemento simmetrico del vettore.
Linea 2530: pone nell'elemento simmetrico il valore di T.
Linea 2540: chiude il loop.
Linea 2550 torna dalla routine.

SEZIONE 9.
Linee 2600-2620: trasferisce il vettore BY nel vettore B2 e lo azzera.
Linea 2630: per ogni bit degli elementi del vettore esegue fino a 2660 (indice I).
Linea 2640: per ogni elemento del vettore esegue fino a 2660 (indice J).
Linea 2645-2650: pone il bit considerato dell'elemento considerato uguale al bit j-esimo dell'elemento simmetrico all'i-esimo.
Linea 2660: chiude i due loop.
Linea 2670: torna dalla routine.

SEZIONE 10.
Linee 3000-3020: chiede il numero (D-CODE) del carattere da correggere, accetta un numero tra 0 e 255 e lo pone nella variabile NC%.
Linea 3030: pulisce lo schermo, scrive centrato in alto PREMI F1 PER USCIRE e pone il cursore al colore blu.
Linea 3040: sposta il cursore più in basso di 8 linee.
Linea 3050: per 8 volte esegue fino alla linea 3100.
Linea 3060: sposta il cursore a destra di 16 posizioni.
Linee 3070-3100: scrive + se il bit corrispondente del carattere considerato contiene 0, uno spazio in campo inverso se contiene 1.
Linea 3110: salta alla routine di programmazione del carattere.

SEZIONE 11.
Linea 4000: pulisce lo schermo e scrive PREMI F1 PER TORNARE.
Linee 4005-4006: fornisce istruzioni all'utente.
Linea 4010: attende circa due secondi.
Linea 4020: cancella la scritta.
Linea 4030 pone la mappa dei caratteri in 12288 (3000H).
Linea 4040: accetta un carattere dalla tastiera.

Linea 4045-4055: se il tasto premuto è F1 riassume la ROM come mappa dei caratteri, pone il cursore al blu chiaro (è infatti possibile vedere i caratteri colorati come si preferisce) e torna dalla routine.
Linea 4060: scrive il carattere ricevuto dalla tastiera.
Linea 4070: torna a ricevere il prossimo carattere.

SEZIONE 12:
Linea 5000: pulisce lo schermo.
Linea 5010: chiede se si vuole salvare su disco o su nastro.
Linea 5015: accetta il carattere D o N.
Linea 5020: porta il cursore più giù di una linea.
Linea 5030: chiede il numero di set che si vuole salvare, NS%, per poter far stare su disco più di un set.
Linea 5040:porta il cursore più giù di una linea.
Linee 5050-5055: chiede il numero di caratteri che si vogliono salvare.
Linea 5060: se la risposta è maggiore di 512 caratteri (2 set completi) torna a porre la domanda.
Linea 5070: se si vuole salvare su nastro salta alla linea 5500.
Linee 5080-5082: apre il file su disco ed esegue la routine di errore.
Linee 5085-5087: se si verifica errore attende circa 2 secondi per far leggere la scritta, chiude il file e il canale di comando e torna dalla routine.
Linee 5090-5110: scrive su disco il numero di caratteri da salvare e i caratteri.
Linea 5120: chiude il file e il canale di comando.
Linea 5130: torna dalla routine.
Linea 5500: apre il file su nastro.
Linee 5510-5530: scrive su nastro il numero di caratteri da salvare e i caratteri.
Linea 5540: chiude il file su nastro.
Linea 5550 torna dalla routine.

SEZIONE 13.
Linea 6000: pulisce lo schermo.
Linea 6010: chiede se si vuole caricare da disco o da nastro.
Linea 6015: accetta il carattere D o N.
Linea 6020: porta il cursore più giù di una linea.
Linea 6030: chiede il numero del set che si vuole caricare.
Linea 6070: se si vuole caricare da nastro salta alla linea 6500.
Linee 6080-6082: apre il file su disco ed esegue la routine di errore.
Linee 6085-6087: se si verifica errore attende circa 2 secondi per far leggere la scritta, chiude il file e il canale di comando e torna dalla routine.
Linee 6090-6110 legge dal disco il numero di caratteri da caricare e i caratteri.
Linea 6120: chiude il file e il canale di comando.
Linea 6130: torna dalla routine.
Linea 6500: apre il file su nastro.

Linee 6510-6530: carica da nastro il numero di caratteri da caricare e i caratteri.
Linea 6540: chiude il file su nastro.
Linea 6550: torna dalla routine.

SEZIONE 14.
Linea 7000: porta il cursore più giù di una linea e ti chiede se sei sicuro di voler abbandonare il programma.
Linea 7010:accetta un carattere dalla tastiera.
Linea 7020: se la risposta è S allora salta alla routine di inizializzazione del BASIC.
Linea 7030: altrimenti torna dalla routine.

SEZIONE 15.
Linee 9000-9005: abbassa il cursore di una linea, apre il canale di comando, riceve dal disco il messaggio e lo scrive sul video.
Linea 9010: torna dalla routine.

Con questo programma è quindi possibile creare in memoria nuovi caratteri e salvarli su disco o cassetta. Se, in un programma che usa caratteri definiti, vuoi evitare di caricare i dati relativi ai caratteri da cassetta, e preferisci che siano già nel programma sotto forma di linee DATA, puoi usare il programma GRAF6, che converte i byte contenuti dalla cella 12288 (3000H) in poi in linee DATA.

```
10 REM GRAF6
20 INPUT"FINO A CHE CARATTERE ";C%
30 PRINT:I=0
40 IFC%>255THENPRINT"MAX 255":PRINT:GOTO20
50 PRINTCHR$(147);
60 N=1000+I*10:GOSUB160:PRINTN$"DATA";
70 FORJ=0TO7
80 N=PEEK(12288+I*8+J):GOSUB160:PRINTN$",";
90 NEXT
100 PRINTCHR$(20)
110 PRINT"I="I+1":C%="C%;
120 IFI<C%THENPRINT":GOTO50":GOTO140
130 PRINT":GOTO170"
140 POKE198,3:POKE631,19:POKE632,13
150 POKE633,13:END
160 N$=STR$(N):N$=RIGHT$(N$,LEN(N$)-1):RETURN
170 POKE198,3:POKE631,19:POKE632,13:POKE633,13
180 K=K+10:PRINTCHR$(147)K
190 PRINT"K="K":IFK<170THEN170"
```

GRAF6 che può essere usato anche per convertire in linee DATA programmi scritti in linguaggio macchina o altri tipi di dati presenti in memoria, si basa sul fatto che il COMMODORE 64, quando esce da un programma, scrive i dati che trova nel buffer della tastiera (631-640 (277H-280H)) e si comporta esattamente come se fossero stati premuti dall'utente. Osserva bene come funziona questo piccolo TRUCCO che verrà ripetuto anche in altri programmi di questo libro.

COMMENTO A GRAF6.
Linea 10: pone i puntatori di fine memoria a 12288 (3000H) per non cancellare, con le variabili, i dati posti da quella locazione in poi.
Linea 20: chiede fino a che carattere vuoi convertire in linea DATA. Il programma convertirà 8*(NC+1) byte a partire dall'indirizzo 12288.
Linee 30-40: scende di una linea e, se il numero di caratteri supera il massimo consentito per un set (256) torna alla linea 20. Questa linea può essere tolta quando si vuole convertire più dati.
Linea 50: pulisce lo schermo.
Linea 60: pone il numero che avrà la prossima linea DATA uguale a 1000+I*10, trasforma N in una stringa (appoggiandosi a una routine in 140), scrive la stringa contenente N e la parola BASIC DATA. Si può scegliere da quale linea partire con le linee DATA (sostituendo a 1000 il numero di linea desiderato) e con che passo incrementare il numero di linea (sostituendo a 10 il passo desiderato).
Linee 70-90: per gli 8 byte di ogni carattere pone in N il contenuto del byte interessato, lo trasforma in una stringa usando la stessa routine usata prima, scrive la stringa e una virgola.
Linea 100: cancella l'ultima virgola.
Linee 110-120: scrive sul video dei comandi BASIC che incrementeranno l'indice I e rimetteranno in memoria il valore di C (le variabili vengono infatti perse ogni volta che si introduce una nuova linea del programma). Se non è stato considerato l'ultimo carattere la linea 120 scrive anche GOTO 50.
Linea 130: se è stato considerato l'ultimo carattere viene scritto GOTO 170.
Linee 140-150: pone 3 nel byte 198 (il byte 198 indica il numero di caratteri validi del buffer di tastiera), pone nel buffer di tastiera il codice ASCII di HOME e due volte il codice ASCII di RETURN e esce dal programma. A questo punto il calcolatore trova nel buffer di tastiera HOME e quindi porta il cursore sulla linea DATA, un RETURN, e quindi introduce nel programma la linea DATA portando il cursore sulla seconda linea scritta dal programma; trova il secondo RETURN ed esegue quindi i comandi scritti sulla linea, aggiorna cioè le variabili e salta a 50 o a 170.
Linea 160: è la routine che trasforma il numero N nella corrispondente stringa N$ usando la funzione STR$ e scartando il carattere che contiene il segno.

Linee 170-190: usando lo stesso metodo descritto sopra cancella le linee da 10 a 170 compresa. Restano nel programma le linee 180 e 190; per evitare questo, puoi trasformare le 3 linee, da 170 a 190, in un unica linea con i separatori due punti (noi non l'abbiamo potuto fare per esigenze tipografiche).

2.1.3 Caratteri a sfondo programmabile

Come abbiamo già visto, quando scrivi un carattere puoi sceglierne il colore, selezionando il colore del cursore o scrivendo nella memoria del colore il codice desiderato. In questo modo selezioni il colore dell'INCHIOSTRO con cui il COMMODORE 64 scrive il carattere, ma quello della CARTA (lo sfondo) rimane sempre lo stesso. Il VIC II ha la capacità di poter leggere diversi colori di sfondo per ogni carattere quando viene posto in modo SFONDO PROGRAMMABILE. In questo modo, infatti, quando il VIC II legge dalla mappa video il codice di un carattere considera i 6 bit meno significativi come codice del carattere, e i due bit rimanenti come codice del colore dello sfondo. Hai così a disposizione un set di soli 64 (2 elevato 6) caratteri, ma puoi scegliere per ciascuno di essi un colore di sfondo su 4 possibili. Puoi naturalmente scegliere a tuo piacere questi 4 colori tra i 16 a disposizione. Con il modo a sfondo programmabile potrai fare delle bellissime maschere o dei coloratissimi quadri grafici. I caratteri che si perdono con il modo a sfondo programmabile sono i caratteri in campo inverso in entrambi i set, i caratteri grafici nel set grafico, le maiuscole e i caratteri grafici nel set commerciale. I 4 colori che vengono scelti come possibili sfondi vanno posti nel registro 33 (21H), lo stesso registro del colore dello schermo, nel registro 34 (22H), nel registro 35 (23H) e nel registro 36 (24H) che rispondono agli indirizzi 53281, 53282, 53283, 53284.

- Il primo colore viene dato come sfondo ai caratteri che hanno nei due bit più significativi 00.
- Il secondo colore viene dato come sfondo ai caratteri che hanno nei due bit più significativi 01.
- Il terzo colore viene dato come sfondo ai caratteri che hanno nei due bit più significativi 10.
- Il quarto colore viene dato come sfondo ai caratteri che hanno nei due bit più significativi 11.

Ricorda che il bit più significativo è a 1 per i caratteri in campo inverso. Nel modo a sfondo programmabile premendo A si ottiene sul video una normale A, premendo SHIFT-A si ottiene una A su un campo di colore 2, una A in campo inverso diventa una A su campo di colore 3, e uno SHIFT-A in campo inverso diventa una A su campo di colore 4. Questa regola vale per tutte le lettere dell' alfabeto, la freccia in alto (elevato a) e lo spazio. Non vale, purtroppo per i numeri e i segni di punteggiatura; per questi caratteri riportiamo la Tabella 2.1 per trovare le corrispondenze. Per entrare in modo a sfondo programmabile basta porre a 1 il bit di posizione 6 del

registro 17 del VIC II che risponde all'indirizzo 53265 (D011H). L'istruzione è la seguente:
POKE 53265,PEEK(53265) OR 64
per uscire dal modo a sfondo programmabile bisogna scrivere:
POKE 53265,PEEK(53265) AND 191.
Come prova di programmazione dello sfondo dei caratteri, segue il programma GRAF7.

```
1 REM GRAF7
10 PRINTCHR$(147):FORI=1TO8:PRINTCHR$(17):NEXT
20 POKE53265,PEEK(53265)OR64
30 POKE53280,0
40 POKE53281,0
50 POKE53282,12
60 POKE53283,14
70 POKE53284,2
80 PRINTCHR$(147);:FORI=1TO5:PRINT:NEXT
90 PRINTTAB(13)CHR$(30)"COMMODORE 64"
100 PRINT:PRINT
101 PRINTTAB(9)CHR$(28)"                    "
103 PRINTTAB(9)" ┐└┘│ ♠\\♠ ─\ ┐└┘x♠ "
105 PRINTTAB(9)"                    "
110 PRINT
111 PRINTTAB(13)CHR$(31)CHR$(18)"            "
113 PRINTTAB(13)CHR$(18)" CARATTERI "
115 PRINTTAB(13)CHR$(18)"           "
120 PRINT:PRINTTAB(7)CHR$(158)CHR$(18);
121 PRINT"                    "
123 PRINTTAB(7)CHR$(18)" ♠ ♥┌┘/┐┌ ┐└┘│ ♠\\♠│\└ "
125 PRINTTAB(7)CHR$(18);
127 PRINT"                    "
130 GETA$:IFA$=""THEN130
140 POKE53265,PEEK(53265)AND191
```

COMMENTO A GRAF7.
Linea 20: entra nel modo a sfondo programmabile ponendo a 1 il bit di posizione 6 del registro 17 del VIC II.
Linea 30: colore del bordo = nero.
Linea 40: colore dello schermo (primo colore di sfondo) = nero.

Linea 50: secondo colore di sfondo = grigio.
Linea 60: terzo colore di sfondo = blu chiaro.
Linea 70: quarto colore di sfondo = rosso.
Linea 80: pulisce lo schermo e porta il cursore sulla quinta linea.
Linea 90: scrive, al centro, in verde: COMMODORRE 64.
Linee 100-101: scrive 3 righe più in basso, al centro, 20 spazi SHIFTATI e cambia il colore del cursore in rosso.
Linea 103: scrive, al centro la scritta SHIFTATA: PROGRAMMA DI PROVA. Anche gli spazi sono SHIFTATI.
Linea 105: scrive, al centro, 20 spazi SHIFTATI.
Linee 110-111: scrive due righe più in basso al centro, 11 spazi IN CAMPO INVERSO e cambia il colore del cursore in blu.
Linea 113: scrive, al centro la scritta IN CAMPO INVERSO: CARATTERI.
Linea 115: scrive, al centro, 11 spazi IN CAMPO INVERSO.
Linee 120-121: scrive due righe più in basso al centro, 24 spazi IN CAMPO INVERSO e SHIFTATI e cambia il colore del cursore in blu chiaro.
Linea 123: scrive, al centro la scritta IN CAMPO INVERSO e SHIFTATA: A SFONDO PROGRAMMABILE. Anche gli spazi sono SHIFTATI.
Linea 125: scrive, al centro, 24 spazi IN CAMPO INVERSO e SHIFTATI.

Ti proponiamo ora il programma GRAF8,leggermente più complicato che mostra il modo a sfondo programmabile applicato ai caratteri programmabili: in questo esempio DISEGNAMO alcuni caratteri e li disponiamo sul video più volte con sfondi diversi:

```
1 REM GRAF8
10 POKE56,48:POKE55,0:CLR:DM=55296-1024
20 FORI=12288TO12479
30 READA
40 POKEI,A
50 NEXT
55 FORI=0TO7:POKE12544+I,0:NEXT
60 PRINTCHR$(147)
70 POKE53280,1:POKE53281,14
80 POKE53272,(PEEK(53272)AND240)+12
90 POKE53265,PEEK(53265)OR64
100 POKE53282,5
110 POKE53283,8
120 POKE53284,7
130 FORI=1544TO1703:POKEI,96:NEXT
140 FORI=1704TO1863:POKEI,160:NEXT
```

```
150 FORI=1864TO2023:POKEI,96:NEXT
160 FORI=0TO2:FORJ=0TO2:POKE1501+J+I*40,224
165 NEXT:NEXT
170 PRINTCHR$(19);:FORI=0TO7:PRINTCHR$(17);:NEXT
175 PRINTCHR$(157)CHR$(28)"V";
180 A$="VWWW":FORI=0TO2
190 FORJ=0TO37-I:PRINTCHR$(29);:NEXT
200 PRINTLEFT$(A$,I+2);
210 NEXT
220 OG=1540:T=1:GOSUB300
240 OG=1630:T=0:GOSUB300
250 OG=1490:T=0:GOSUB300
260 OG=1800:T=1:GOSUB300
270 GOTO270
300 IFTTHEN420
310 POKEOG,PEEK(OG)-32:POKEOG+DM,1:OG=OG+1
320 POKEOG,PEEK(OG)-32+1:POKEOG+DM,1:OG=OG+39
330 POKEOG,PEEK(OG)-32+2:POKEOG+DM,6:OG=OG+1
340 POKEOG,PEEK(OG)-32+3:POKEOG+DM,6:OG=OG+38
350 POKEOG,PEEK(OG)-32+6:POKEOG+DM,6:OG=OG+1
360 POKEOG,PEEK(OG)-32+4:POKEOG+DM,6:OG=OG+1
370 POKEOG,PEEK(OG)-32+5:POKEOG+DM,6:OG=OG+39
380 POKEOG,PEEK(OG)-32+7:POKEOG+DM,1:OG=OG+1
390 POKEOG,PEEK(OG)-32+8:POKEOG+DM,1:OG=OG+39
400 POKEOG,PEEK(OG)-32+9:POKEOG+DM,1:OG=OG+1
410 POKEOG,PEEK(OG)-32+10:POKEOG+DM,1:RETURN
420 POKEOG,PEEK(OG)-32+12:POKEOG+DM,1:OG=OG+1
430 POKEOG,PEEK(OG)-32+11:POKEOG+DM,1:OG=OG+39
440 POKEOG,PEEK(OG)-32+14:POKEOG+DM,6:OG=OG+1
450 POKEOG,PEEK(OG)-32+13:POKEOG+DM,6:OG=OG+39
460 POKEOG,PEEK(OG)-32+16:POKEOG+DM,6:OG=OG+1
470 POKEOG,PEEK(OG)-32+15:POKEOG+DM,6:OG=OG+1
480 POKEOG,PEEK(OG)-32+17:POKEOG+DM,6:OG=OG+38
490 POKEOG,PEEK(OG)-32+19:POKEOG+DM,1:OG=OG+1
500 POKEOG,PEEK(OG)-32+18:POKEOG+DM,1:OG=OG+39
510 POKEOG,PEEK(OG)-32+21:POKEOG+DM,1:OG=OG+1
520 POKEOG,PEEK(OG)-32+20:POKEOG+DM,1:RETURN
1000 DATA0,0,1,3,7,15,31,31
1010 DATA0,96,240,240,224,192,192,224
1020 DATA31,63,63,127,127,63,31,7
1030 DATA224,54,191,255,246,240,224,0
1040 DATA7,31,62,125,251,247,207,159
```

```
1050 DATA0,176,216,220,238,239,231,195
1060 DATA0,0,0,0,1,7,7,15
1070 DATA31,31,15,23,27,29,30,60
1080 DATA192,128,128,225,243,119,127,62
1090 DATA126,255,126,0,0,0,0,0
1100 DATA28,0,0,0,0,0,0,0
1110 DATA0,0,128,192,224,240,248,248
1120 DATA0,6,15,15,7,3,3,7
1130 DATA248,252,252,254,254,252,248,224
1140 DATA7,108,253,255,111,15,7,0
1150 DATA224,248,124,190,223,239,243,249
1160 DATA0,13,27,59,119,247,231,195
1170 DATA0,0,0,0,128,224,224,240
1180 DATA248,248,240,232,216,184,120,60
1190 DATA3,1,1,135,207,238,254,124
1200 DATA126,255,126,0,0,0,0,0
1210 DATA56,0,0,0,0,0,0,0
1220 DATA1,3,7,15,31,63,127,255
1230 DATA255,255,255,255,255,255,255,255
```

GRAF8 crea 22 caratteri necessari a disegnare dei puffi e, grazie a una subroutine, li pone in un qualsiasi punto di uno schermo variopinto.

COMMENTO A GRAF8.
Linea 10: pone la fine della memoria a 12288 (3000H) per proteggere i caratteri dalle variabili e pone nella variabile DM (differenza mappe) la differenza tra il primo indirizzo della mappa del colore e il primo indirizzo della mappa video.
Linee 20-50: trasferisce i dati delle immagini in memoria.
Linea 55: pone 8 zeri nel carattere 32 (lo spazio).
Linea 60: pulisce lo schermo.
Linea 70: bordo nero e schermo (colore sfondo 1) azzurro.
Linea 80: avvisa il VIC II che la mappa dei caratteri è in 12288 (3000H).
Linea 90: entra in modo a sfondo programmabile.
Linea 100: colore sfondo 2 = verde.
Linea 110: colore sfondo 3 = giallo.
Linea 120: colore sfondo 4 = marrone.
Linea 130: disegna una striscia verde usando degli spazi SHIFTATI (codice schermo = 96).
Linea 140: disegna una striscia marrone usando degli spazi IN CAMPO INVERSO (codice schermo = 160).

Linea 150: disegna una seconda striscia verde usando spazi SHIFTATI.
Linea 160: disegna una casetta gialla usando spazi IN CAMPO INVERSO e SHIFTATI.
Linee 170-210: disegna il tetto della casa usando dei caratteri appositamente programmati. Nota che in questa zona del video non potrai mettere dei puffi perchè il tetto non è stato ottenuto con degli spazi a sfondo colorato come nel caso delle strisce colorate o della casetta.
Linea 220: disegna un puffo di tipo 1 (che guarda a sinistra), il cui angolo in alto a sinistra verrà posto nel byte della memoria video 1540 usando la subroutine in 300.
Linea 240: disegna un puffo di tipo 0 (che guarda a destra), il cui angolo in alto a sinistra verrà posto nel byte della memoria video 1630 usando la subroutine in 300.
Linea 250: disegna un puffo di tipo 0 (che guarda a destra), il cui angolo in alto a sinistra verrà posto nel byte della memoria video 1490 usando la subroutine in 300.
Linea 260: disegna un puffo di tipo 1 (che guarda a sinistra), il cui angolo in alto a sinistra verrà posto nel byte della memoria video 1800 usando la subroutine in 300.
Linea 270: ferma il programma (per uscirne dovrai premere STOP/RESTORE)
Linea 300: salta se T < > 0 quindi se il puffo è di tipo 1.
Linee 310-410: ogni linea disegna uno degli 11 caratteri che compongono un puffo di tipo 0 e pone nel corrispondente byte del colore il colore adatto. Per disegnare un carattere in un byte viene sottratto dal contenuto del byte il numero 32 (codice dello spazio) e sommato il codice desiderato: in questo modo i due bit più significativi rimangono inalterati e quindi non cambia il colore dello sfondo.
Linee 420-520: ogni linea disegna uno degli 11 caratteri che compongono un puffo di tipo 1 e pone nel corrispondente byte del colore il colore adatto.
Linee 1000-1230: dati relativi ai caratteri dei puffi e del tetto della casetta.

TABELLA DI CORRISPONDENZA				
SIMB.	TASTI DA PREMERE			
	COLORE 1	COLORE 2	COLORE 3	COLORE 4
@	@	SHIFT+*	RVS+@	RVS+SHIFT+*
A	A	SHIFT+A	RVS+A	RVS+SHIFT+A
B	B	SHIFT+B	RVS+B	RVS+SHIFT+B
C	C	SHIFT+C	RVS+C	RVS+SHIFT+C
D	D	SHIFT+D	RVS+D	RVS+SHIFT+D
E	E	SHIFT+E	RVS+E	RVS+SHIFT+E
F	F	SHIFT+F	RVS+F	RVS+SHIFT+F
G	G	SHIFT+G	RVS+G	RVS+SHIFT+G
H	H	SHIFT+H	RVS+H	RVS+SHIFT+H
I	I	SHIFT+I	RVS+I	RVS+SHIFT+I
J	J	SHIFT+J	RVS+J	RVS+SHIFT+J
K	K	SHIFT+K	RVS+K	RVS+SHIFT+K
L	L	SHIFT+L	RVS+L	RVS+SHIFT+L
M	M	SHIFT+M	RVS+M	RVS+SHIFT+M
N	N	SHIFT+N	RVS+N	RVS+SHIFT+N
O	O	SHIFT+O	RVS+O	RVS+SHIFT+O

Tabella 2.1 Corrispondenze TASTI-CODICE COLORE per caratteri a sfondo programmabile

TABELLA DI CORRISPONDENZA				
SIMB.	TASTI DA PREMERE			
	COLORE 1	COLORE 2	COLORE 3	COLORE 4
P	P	SHIFT+P	RVS+P	RVS+SHIFT+P
Q	Q	SHIFT+Q	RVS+Q	RVS+SHIFT+Q
R	R	SHIFT+R	RVS+R	RVS+SHIFT+R
S	S	SHIFT+S	RVS+S	RVS+SHIFT+S
T	T	SHIFT+T	RVS+T	RVS+SHIFT+T
U	U	SHIFT+U	RVS+U	RVS+SHIFT+U
V	V	SHIFT+V	RVS+V	RVS+SHIFT+V
W	W	SHIFT+W	RVS+W	RVS+SHIFT+W
X	X	SHIFT+X	RVS+X	RVS+SHIFT+X
Y	Y	SHIFT+Y	RVS+Y	RVS+SHIFT+Y
Z	Z	SHIFT+Z	RVS+Z	RVS+SHIFT+Z
[	[	SHIFT++	RVS+[	RVS+SHIFT++
£	£	CBM+-	RVS+£	RVS+CBM+-
]	]	SHIFT+-	RVS+]	RVS+SHIFT+-
↑	↑	SHIFT+↑	RVS+↑	RVS+SHIFT+↑
←	←	CBM+*	RVS+←	RVS+CBM+*

Tabella 2.1 Corrispondenze TASTI-CODICE COLORE per caratteri a sfondo programmabile (continuazione)

TABELLA DI CORRISPONDENZA				
SIMB.	TASTI DA PREMERE			
	COLORE 1	COLORE 2	COLORE 3	COLORE 4
SP	SP	SHIFT+SP	RVS+SP	RVS+SHIFT+SP
!	!	CBM+K	RVS+!	RVS+CBM+K
"	"	CBM+I	RVS+"	RVS+CBM+I
#	#	CBM+T	RVS+#	RVS+CBM+T
$	$	CBM+@	RVS+$	RVS+CBM+@
%	%	CBM+G	RVS+%	RVS+CBM+G
&	&	CBM++	RVS+&	RVS+CBM++
'	'	CBM+M	RVS+'	RVS+CBM+M
(	(	CBM+£	RVS+(	RVS+CBM+£
)	)	SHIFT+£	RVS+)	RVS+SHIFT+£
*	*	CBM+N	RVS+*	RVS+CBM+N
+	+	CBM+Q	RVS++	RVS+CBM+Q
,	,	CBM+D	RVS+,	RVS+CBM+D
-	-	CBM+Z	RVS+-	RVS+CBM+Z
.	.	CBM+S	RVS+.	RVS+CBM+S
/	/	CBM+P	RVS+/	RVS+CBM+P

Tabella 2.1 Corrispondenze TASTI-CODICE COLORE per caratteri a sfondo programmabile (continuazione)

TABELLA DI CORRISPONDENZA				
SIMB.	TASTI DA PREMERE			
	COLORE 1	COLORE 2	COLORE 3	COLORE 4
0	0	CBM+A	RVS+0	RVS+CBM+A
1	1	CBM+E	RVS+1	RVS+CBM+E
2	2	CBM+R	RVS+2	RVS+CBM+R
3	3	CBM+W	RVS+3	RVS+CBM+W
4	4	CBM+H	RVS+4	RVS+CBM+H
5	5	CBM+J	RVS+5	RVS+CBM+J
6	6	CBM+L	RVS+6	RVS+CBM+L
7	7	CBM+Y	RVS+7	RVS+CBM+Y
8	8	CBM+U	RVS+8	RVS+CBM+U
9	9	CBM+O	RVS+9	RVS+CBM+O
:	:	SHIFT+@	RVS+:	RVS+SHIFT+@
;	;	CBM+F	RVS+;	RVS+CBM+F
<	<	CBM+C	RVS+<	RVS+CBM+C
=	=	CBM+X	RVS+=	RVS+CBM+X
>	>	CBM+V	RVS+>	RVS+CBM+V
?	?	CBM+B	RVS+?	RVS+CBM+B

Tabella 2.1 Corrispondenze TASTI-CODICE COLORE per caratteri a sfondo programmabile (continuazione)

2.1.4 Caratteri multicolore

Questo paragrafo tratta dell'ultimo modo che hai a disposizione per rappresentare i caratteri con il COMMODORE 64: il modo multicolore. Nel modo alta risoluzione (il modo normale) puoi colorare ogni puntino chè forma un carattere agendo sul bit ad esso corrispondente. Ogni punto può valere 0 o 1, cioè essere colorato o no. Nel modo multicolore ad ogni punto che forma il carattere corrispondono 2 bit: in questo modo ogni punto può essere di un colore scelto tra 4 (i colori di sfondo 1, 2, 3 e il colore proprio della locazione video). L'unica cosa che viene persa, passando dal modo ad alta risoluzione al modo multicolore, è la risoluzione orizzontale: i caratteri non corrispondono più a matrici 8*8, ma corrispondono a matrici 4*8. Per entrare in modo multicolore bisogna porre a 1 il bit di posizione 4 del registro 22 del VIC II (indirizzo 53270 (D016H)): bisogna perciò eseguire il seguente comando:
POKE 53270,PEEK(53270) OR 16.
Per tornare in modo alta risoluzione dovrai eseguire il comando:
POKE53270,PEEK(53270) AND 239.
Per chiarire le idee vediamo come il VIC II legge il carattere A in modo multicolore:
Ecco il contenuto degli 8 byte relativi alla A:

```
0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 0 0 0 0 0 0 0
```

Il VIC II, nel modo multicolore li legge, come 4 punti colore per riga, così:

00 01 10 00	cioè come:	A B C A
00 11 11 00		A D D A
01 10 01 10		B C B C
01 11 11 10		B D D C
01 10 01 10		B C B C
01 10 01 10		B C B C
01 10 01 10		B C B C
00 00 00 00		A A A A

dove A (da non confondere con il carattere A preso come spunto), B, C e D sono dei punti colore e:
A è del colore dello schermo
B è del colore di sfondo 2
C è del colore di sfondo 3
D è del colore proprio del byte.

I primi 3 colori possono essere scelti tra i 16 del COMMODORE 64, mentre il quarto può essere solo uno dei primi 8. Questa piccola limitazione permette, in compenso, di visualizzare contemporaneamente caratteri in multicolore e ad alta risoluzione, il VIC II infatti visualizza il carattere come multicolore solo se il byte corrispondente della mappa del colore ha il bit di posizione 3 a 1 e assume come codice del colore il numero formato dai 3 bit meno significativi.
Ecco GRAF9, come esempio di programmazione di un carattere multicolore:

```
1 REM GRAF9
10 PRINTCHR$(147)CHR$(150)
20 POKE53270,PEEK(53270)OR16
30 FORI=0TO7:READA:POKE12288+I,A:NEXT
40 FORI=0TO7:POKE12288+32*8+I,0:NEXT
50 POKE53280,0
60 POKE53281,0
70 POKE53282,5
80 POKE53283,6
90 POKE53272,(PEEK(53272)AND240)+12
100 PRINT"@"
110 GETA$:IFA$=""THEN110
120 POKE53270,PEEK(53270)AND239
130 POKE53272,(PEEK(53272)AND240)+5
140 PRINTCHR$(154)
1000 DATA165,165,165,165,240,240,240,240
```

COMMENTO A GRAF9.
Linea 10: pulisce lo schermo e pone il colore del cursore rosa (codice 10). Il colore di un carattere multicolore scritto con il codice 10 sarà quindi del colore 10-8=2 (rosso).
Linea 20: entra in modo multicolore.
Linea 30: legge gli 8 dati relativi al carattere chiocciola e li pone a partire da 12288 (3000H).

Linea 40: azzera gli 8 byte relativi al carattere di codice 32 (lo spazio).
Linea 50: colore del bordo = nero.
Linea 60: colore dello schermo = nero.
Linea 70: colore di sfondo 2 = verde.
Linea 80: colore di sfondo 3 = blu.
Linea 90: pone la mappa dei caratteri in 12288 (3000H).
Linea 100: stampa il carattere di D-CODE uguale a 0 (la chiocciola).
Linea 110: attende che sia premuto un tasto.
Linea 120: torna in modo alta risoluzione.
Linea 130: riattiva i caratteri dalla ROM.
Linea 140: colore del cursore blu chiaro.
Linea 1000: contiene i dati relativi al carattere.

Questo esempio mostra chiaramente come il VIC II sia capace di porre 4 colori diversi nella matrice di un solo carattere, ma il risultato non è certamente dei più pittoreschi. Prova quindi il programma GRAF10, il cui scopo è quello di disegnare dei coloratissimi omini spaziali.

```
1 REM GRAF10
10 PRINTCHR$(147)CHR$(155)
20 POKE53270,PEEK(53270)OR16
30 FORI=0TO47:READA:POKE12288+I,A:NEXT
40 FORI=0TO7:POKE12288+32*8+I,0:NEXT
50 POKE53280,0
60 POKE53281,0
70 POKE53282,2
80 POKE53283,6
90 POKE53272,(PEEK(53272)AND240)+12
100 FORI=0TO9:PRINT:NEXT
110 PRINT" @A @A @A @A @A @A @A";
115 PRINT" @A @A @A @A @A @A"
120 PRINT" BC BC BC BC BC BC BC";
125 PRINT" BC BC BC BC BC BC"
130 PRINT" DE DE DE DE DE DE DE";
135 PRINT" DE DE DE DE DE DE"
140 GETA$:IFA$=""THEN140
150 POKE53270,PEEK(53270)AND239
160 POKE53272,(PEEK(53272)AND240)+5
170 PRINTCHR$(154)
1000 DATA192,192,49,53,21,81,65,85
```

```
1010 DATA3,3,76,92,84,69,65,85
1020 DATA84,85,22,5,3,5,21,85
1030 DATA21,85,148,80,192,80,84,85
1040 DATA85,85,21,5,1,34,42,10
1050 DATA85,85,84,80,64,136,168,160
```

COMMENTO A GRAF10.
Linea 10: pulisce lo schermo e pone il colore del cursore a grigio chiaro (codice 15). Il colore di un carattere multicolore scritto col codice 15 sarà quindi di codice 15-8=7 (giallo).
Linea 20: entra in modo multicolore.
Linea 30: legge i 48 dati relativi ai 6 caratteri e li pone a partire da 12288 (3000H).
Linea 40: azzera gli 8 byte relativi al carattere di codice 32 (lo spazio).
Linea 50: colore del bordo = nero.
Linea 60: colore dello schermo = nero.
Linea 70: colore di sfondo 2 = rosso.
Linea 80: colore di sfondo 3 = blu.
Linea 90: pone la mappa dei caratteri in 12288 (3000H).
Linea 100: porta il cursore 10 linee più in basso.
Linea 110: disegna 13 teste.
Linea 120: disegna 13 corpi.
Linea 130: disegna 13 paia di piedi.
Linea 140: attende che sia premuto un tasto.
Linea 150: torna in modo alta risoluzione.
Linea 160: riattiva i caratteri della ROM.
Linea 170: colore del cursore blu chiaro.
Linea 1000: contiene i dati relativi al carattere.

Come per i caratteri creati nel modo alta risoluzione, è possibile scrivere un programma che aiuti nella creazione dei caratteri multicolore. Riportiamo il listato completo del programma GRAF11 che svolge questa funzione. Il commento è limitato a quelle parti in cui differisce dal programma GRAF5 per la creazione dei caratteri ad alta risoluzione.

```
1 REM GRAF11
10 POKE13*4096+32,0:POKE13*4096+33,0
15 PRINTCHR$(154)CHR$(142)CHR$(8)
20 POKE56,48:POKE55,0:CLR
25 US$=CHR$(147)+"PREMI F1 PER USCIRE"
```

```
30 DIMC(15):FORI=0TO15:READC(I):NEXT
40 GOSUB8000
100 PRINTCHR$(147)
110 PRINT"OPZIONI :":PRINT:PRINT
120 PRINT"1 CREA CARATTERE":PRINT
130 PRINT"2 MEMORIZZA CARATTERE":PRINT
140 PRINT"3 CORREGGE CARATTERE":PRINT
150 PRINT"4 MOSTRA CARATTERI":PRINT
160 PRINT"5 SALVA SET DI CARATTERI":PRINT
170 PRINT"6 CARICA SET DI CARATTERI":PRINT
180 PRINT"7 SCELTA COLORI":PRINT
185 PRINT"8 FINE"
190 GETA$:IFVAL(A$)=0THEN190
195 I=VAL(A$)
197 IFI=8THENGOSUB7000
200 ONIGOSUB1000,2000,3000,4000,5000,6000,8000
210 GOTO100
1000 GOSUB1500
1005 PRINTUS$:FORI=1TO200:NEXT
1010 POKE53281,-11*(CL=0):PRINTCHR$(C(CL));
1015 FORI=1TO8:PRINTCHR$(17);:NEXTI
1020 FORI=1TO8
1030 FORJ=1TO16:PRINTCHR$(32);:NEXT
1040 PRINTCHR$(18);
1045 FORJ=1TO8:PRINTCHR$(32);:NEXTJ
1050 PRINT:NEXTI
1060 PC=1400:XC=0:YC=0
1070 POKEPC,86:POKEPC+1,86
1080 GETA$:IFA$=""THEN1080
1090 A=ASC(A$):SP=0
1100 IFA=29ANDXC<6THENXC=XC+2:PC=PC+2:SP=2
1110 IFA=157ANDXC>0THENXC=XC-2:PC=PC-2:SP=-2
1120 IFA=17ANDYC<7THENYC=YC+1:PC=PC+40:SP=40
1130 IFA=145ANDYC>0THENYC=YC-1:PC=PC-40:SP=-40
1140 IFA=134THENPOKEPC+54272,CC:POKEPC+54273,CC
1141 IFA=135THENPOKEPC+54272,PEEK(53282)
1142 IFA=135THENPOKEPC+54273,PEEK(53282)
1143 IFA=136THENPOKEPC+54272,PEEK(53283)
1144 IFA=136THENPOKEPC+54273,PEEK(53283)
1150 IFA=20THENPOKEPC+54272,CL:POKEPC+54273,CL
1160 IFA=133THENPRINTCHR$(154):POKEPC,160
1165 IFA=133THENPOKEPC+1,160:POKE53281,0
```

```
1167 IFA=133THENGOTO1180
1170 POKEPC-SP,160:POKEPC-SP+1,160:GOTO1070
1180 FORI=0TO7
1190 BY(I)=0
1200 FORJ=0TO6STEP2
1205 P1=PEEK(55672+40*I+6-J)AND15
1207 P2=(PEEK(53282)AND15)
1208 P3=(PEEK(53283)AND15)
1210 IFP1=CCTHENBY(I)=BY(I)+2↑J+2↑(J+1)
1211 IFP1=P2THENBY(I)=BY(I)+2↑J
1212 IFP1=P3THENBY(I)=BY(I)+2↑(J+1)
1220 NEXTJ:NEXTI
1230 RETURN
1500 PRINTCHR$(147)
1510 PRINT:PRINT:PRINT:PRINT
1520 FORI=0TO7
1530 PRINTTAB(14)CHR$(C(I))CHR$(18)"        ";
1535 PRINTCHR$(154)CHR$(146)" ="I
1540 NEXT
1550 PRINTCHR$(19)TAB(10)"COLORE DI CELLA ";
1555 INPUTCC
1560 RETURN
2000 PRINTCHR$(147)
2010 PRINT"1 NORMALE":PRINT
2030 PRINT"2 SIMMETRIA VERTICALE":PRINT
2040 PRINT"3 SIMMETRIA ORIZZONTALE":PRINT
2060 GETA$:IFA$=""THEN2060
2070 A=VAL(A$):IFA=0ORA>3THEN2060
2080 INPUT"CARATTERE NUMERO ";NC%
2090 IFNC%>511ORNC%<0THEN2080
2100 ONAGOSUB2200,2400,2500
2110 FORI=0TO7:POKE12288+8*NC%+I,BY(I):NEXT
2120 RETURN
2200 RETURN
2400 FORI=0TO7
2410 FORJ=0TO2STEP2
2420 LR=(BY(I)AND2↑J)/2↑J:BY(I)=BY(I)-2↑J*LR
2425 MR=(BY(I)AND2↑(J+1))/2↑(J+1)
2427 BY(I)=BY(I)-2↑(J+1)*MR
2430 ML=(BY(I)AND2↑(7-J))/2↑(7-J)
2432 BY(I)=BY(I)-2↑(7-J)*ML
```

```
2433 LL=(BY(I)AND2↑(6-J))/2↑(6-J)
2434 BY(I)=BY(I)-2↑(6-J)*LL
2440 BY(I)=BY(I)+2↑J*LL+2↑(J+1)*ML
2445 BY(I)=BY(I)+2↑(7-J)*MR+2↑(6-J)*LR
2450 NEXTJ,I
2460 RETURN
2500 FORI=0TO3
2510 T=BY(I)
2520 BY(I)=BY(7-I)
2530 BY(7-I)=T
2540 NEXTI
2550 RETURN
3000 PRINTCHR$(147)
3010 INPUT"CARATTERE DA CORREGGERE ";NC%
3020 IFNC%>511ORNC%<0THEN3010
3030 GOSUB1500
3040 PRINTUS$
3050 FORI=0TO7:FORJ=0TO6STEP2
3055 POKE1400+I*40+J,160:POKE1401+I*40+J,160
3060 FORJ=0TO6STEP2
3070 POKE1400+I*40+J,160
3080 POKE1401+I*40+J,160
3090 A=PEEK(12288+8*NC%+I)AND(2↑(7-J)+2↑(6-J))
3100 IFA=0THENPOKE55672+I*40+J,CL
3110 IFA=0THENPOKE55673+I*40+J,CL
3115 T7=A=2↑(7-J):T6=A=2↑(6-J)
3120 IFT6THENPOKE55672+40*I+J,PEEK(53282)AND15
3130 IFT6THENPOKE55673+40*I+J,PEEK(53282)AND15
3140 IFT7THENPOKE55672+40*I+J,PEEK(53283)AND15
3150 IFT7THENPOKE55673+40*I+J,PEEK(53283)AND15
3160 IFA=3*2↑(6-J)THENPOKE55672+40*I+J,CC
3165 IFA=3*2↑(6-J)THENPOKE55673+40*I+J,CC
3170 NEXT:NEXT
3180 GOTO1060
4000 PRINTUS$
4005 PRINTCHR$(17)"PREMI I TASTI CORRISPONDENTI"
4006 PRINT"AI CARATTERI CHE VUOI VEDERE"
4010 FORI=1TO2000:NEXTI
4020 PRINTCHR$(147);:POKE53281,CL
4030 POKE53272,(PEEK(53272)AND240)+12
4035 POKE53270,PEEK(53270)OR16
```

```
4040 GETA$:IFA$=""THEN4040
4045 F1=A$=CHR$(133)
4050 IFF1THENPOKE53272,(PEEK(53272)AND240)+5
4052 IFF1THENPRINTCHR$(154)
4055 IFF1THENPOKE53281,0
4067 IFF1THENPOKE53270,PEEK(53270)AND239:RETURN
4060 PRINTA$;
4070 GOTO4040
5000 PRINTCHR$(147)
5010 PRINT"DISCO O NASTRO ?"
5015 GETDV$:IFDV$<>"N"ANDDV$<>"D"THEN5015
5020 PRINT
5030 INPUT"SET NUMERO ";NS%
5040 PRINT
5050 PRINT"FINO A CHE CARATTERE VUOI SALVARE "
5055 INPUTNC%
5060 IFNC%>255THEN5040
5070 IFDV$="N"THEN5500
5080 OPEN1,8,2,"@:SET #"+STR$(NS%)+",S,W"
5082 GOSUB9000
5085 IFNNTHENFORI=1TO2000:NEXT:CLOSE1:CLOSE15
5087 IFNNTHENRETURN
5090 PRINT#1,CHR$(NC%);:FORI=0TO(NC%+1)*8-1
5100 PRINT#1,CHR$(PEEK(12288+I));
5110 NEXTI
5120 CLOSE1:CLOSE15
5130 RETURN
5500 OPEN1,1,1,"SET #"+STR$(NS%)
5510 PRINT#1,CHR$(NC%);:FORI=0TO(NC%+1)*8-1
5520 PRINT#1,CHR$(PEEK(12288+I));
5530 NEXTI
5540 CLOSE1
5550 RETURN
6000 PRINTCHR$(147)
6010 PRINT"DISCO O NASTRO ?"
6015 GETDV$:IFDV$<>"N"ANDDV$<>"D"THEN6015
6020 PRINT
6030 INPUT"SET NUMERO ";NS%
6070 IFDV$="N"THEN6500
6080 OPEN1,8,2,"SET #"+STR$(NS%)+",S,R"
6082 GOSUB9000
6085 IFNNTHENFORI=1TO2000:NEXT:CLOSE1:CLOSE15
```

```
6087 IFNNTHENRETURN
6090 GET#1,A$:NC%=ASC(A$):FORI=0TO(NC%+1)*8-1
6100 GET#1,A$:POKE12288+I,ASC(A$+CHR$(0))
6110 NEXTI
6120 CLOSE1:CLOSE15
6130 RETURN
6500 OPEN1,1,0,"SET #"+STR$(NS%)
6510 FORI=0TO(NC+1)%*8-1
6520 GET#1,A$:POKE12288+I,ASC(A$+CHR$(0))
6530 NEXTI
6540 CLOSE1
6550 RETURN
7000 PRINTCHR$(17)"SICURO ?"
7010 GETA$:IFA$=""THEN7010
7020 IFA$="S"THENSYS58260
7030 RETURN
8000 PRINTCHR$(147)
8010 PRINT:PRINT:PRINT:PRINT
8020 FORI=0TO15
8030 PRINTTAB(14)CHR$(C(I))CHR$(18)"        ";
8035 PRINTCHR$(154)CHR$(146)" ="I
8040 NEXT
8050 FORI=1TO3
8060 PRINTCHR$(19)
8070 PRINTTAB(13)"COLORE"I"     ";:FORJ=1TO4
8075 PRINTCHR$(20);:NEXT:INPUTA
8080 IFA<0ORA>15THEN8060
8090 IFI<>1THENPOKE53280+I,A
8100 IFI=1THENCL=A
8110 NEXT
8120 RETURN
9000 PRINT:OPEN15,8,15:INPUT#15,NN,MM$,TT,SS
9005 PRINTNN,MM$,TT,SS
9010 RETURN
10000 DATA144,5,28,159,156,30,31,158
10010 DATA129,149,150,151,152,153,154,155
```

COMMENTO A GRAF 11.

SEZIONE 1.
Aggiunte linee 30 e 40.
Linea 30: dimensiona e riempe il vettore C che conterrà i codici ASCII del colore del cursore. Ad esempio il colore di codice 2 è il rosso, il codice ASCII per ottenere il cursore rosso è 28, il vettore conterrà quindi nella seconda componente il numero 28.
Linea 40: salta alla routine di scelta dei colori (sezione 16).

SEZIONE 2.
E' stata aggiunta un' opzione: la scelta dei colori.

SEZIONE 3.
Si richiede il colore del byte.
La gestione della griglia diventa diversa.

Non si lavora più sui caratteri, ma sui colori, e non ci si sposta più di un solo passo in orizzontale, ma di due. La tecnica con cui viene gestita rimane essenzialmente la stessa.

SEZIONE 4.
Sono state tolte le opzioni REVERSATO, che con i caratteri multicolore non ha senso e RUOTATO DI 90 GRADI, data l'asimmetria dei pixel in multicolore.

SEZIONE 5.
Come prima.

SEZIONE 6.
Eliminata poichè serviva per la memorizzazione in campo inverso.

SEZIONE 7.
Il principio rimane lo stesso ma il secondo ciclo viene svolto a passi di due e vengono scambiate coppie di pixel invece che pixel singoli.

SEZIONE 8.
Come prima.

SEZIONE 9.
Eliminata poichè serviva per la memorizzazione ruotata di 90 gradi.

SEZIONE 10.
Viene richiesto il colore per il byte; come per la sezione 3 la diversità è dovuta al fatto che ora usiamo i colori invece che i caratteri nella griglia.

SEZIONE 11.
Come prima tranne che nella linea 4020 dove viene predisposto il colore richiesto per lo schermo nella routine di scelta dei colori, e nelle linee 4030-4035 dove si entra in modo multicolore.

SEZIONI 12-13-14-15.
Come prima.

SEZIONE 16.
Sono state aggiunte le linee 8000-8120.
Linea 8000: pulisce lo schermo.
Linea 8010: sposta il cursore alla quinta linea.
linea 8020: inizializza un ciclo.
Linea 8030: scrive in colonna 14 sette spazi in campo inverso del colore i-esimo usando il vettore C inizializzato nella linea 30.
Linea 8035: scrive in blu chiaro il codice colore.
Linea 8040: chiude il ciclo.
Linea 8050: inizializza un ciclo.
Linea 8060: pone il cursore in alto a sinistra.
Linea 8070: domanda il colore i-esimo, cancella la risposta precedente, accetta la risposta.
Linea 8080: controlla che il numero corrisponda a un colore, se no torna a ripetere la domanda.
Linea 8090: se il colore richiesto non è quello dello schermo lo pone nel registro appropriato.
Linea 8100: altrimenti nella variabile CL.
Linea 8110: chiude il ciclo.
Linea 8120: torna dalla routine.

CAPITOLO 3

PAGINA GRAFICA

3.1 PAGINA GRAFICA

Poichè il video del COMMODORE 64 è composto da 25 linee di 40 caratteri, e ciascuno di essi e formato da 8*8 punti, lo schermo è composto da ben 64000 punti, disposti in 200 righe di 320 punti. Il calcolatore ha la possibilità di gestire tutti questi punti quando viene posto nel modo PAGINA GRAFICA. Questo modo è molto utile quando vuoi disegnare grafici per applicazioni scientifiche, istogrammi per applicazioni commerciali o quando vuoi progettare giochi. Come per i caratteri, la pagina grafica può essere ad alta risoluzione o multicolore.

3.1.1 Pagina grafica ad alta risoluzione

Per entrare in modo pagina grafica è necessario porre a 1 il bit di posizione 5 del registro 17 del VIC II (indirizzo 53265 (D011H)); cioè eseguire l'istruzione:
POKE 53265,PEEK(53265)OR32.
Quando il COMMODORE 64 è in modo pagina grafica, si comporta come se tutto lo schermo fosse riempito da 1000 caratteri programmabili disposti nel modo indicato nella Tabella 3.1.

```
  0   1   2   3 ............. 39
 40  41  42  43 ............. 79
  .   .   .   . .............  .
  .   .   .   . .............  .
  .   .   .   . .............  .
  .   .   .   . .............  .
960 961 962 963 ............. 999
```

Tabella 3.1 Posizione dei caratteri sullo schermo

L'indirizzo base di questa grande mappa dei caratteri è scritto nello stesso registro in cui è scritto l'indirizzo della normale mappa dei caratteri (vedi Paragrafo 1.3.5). Poichè la disposizione dei caratteri sullo schermo è, nel modo pagina grafica, stabilita dal calcolatore stesso, la memoria video (vedi Paragrafo 1.3.2) viene usata come una più completa memoria del colore (vedi Paragrafo 1.3.3) in cui i 4 bit più significativi danno il colore dei bit posti a 1 e i 4 meno significativi danno il colore dei bit posti a 0. Volendo quindi avere in una certa posizione dello schermo i bit a 0 del colore il cui codice sia A e i bit a 1 di colore il cui codice sia B, bisogna porre nel byte corrispondente a quella posizione il numero B*16+A.
Prova a eseguire le seguenti istruzioni:

```
PRINT CHR$(147)
POKE 53265,PEEK(53265) OR 32
POKE 53272,PEEK(53272) OR 8
```

cioè pulisci lo schermo, entra in modo pagina grafica e poni la mappa in 8192 (2000H).
Come vedi, CHR$(147) non pulisce la pagina grafica ma riempe la mappa video di 32 (ASCII di spazio), in modo pagina grafica quindi agisce sul colore e, più precisamente, pone rossi i bit a 1 e neri i bit a 0 (2*16+0=32, 2=rosso e 0=nero). L'unico modo che abbiamo a disposizione per pulire la pagina grafica è quello di porre a 0 ogni byte della pagina stessa. Le istruzioni da eseguire sono dunque:

```
FOR I = 8192 TO 16191
POKE I,0
NEXT I
```

Purtroppo il BASIC è molto lento nell'eseguire un'operazione di questo tipo e, l'unico modo disponibile per aumentare la velocità è scrivere una piccola routine in linguaggio macchina. Riportiamo ora questa routine, GRAF12, che abbiamo fatto partire dall'indirizzo 49152 (C000H) ma che può essere trasferita senza problemi poichè contiene solamente salti con indirizzi relativi.

Nel seguito compaiono alcuni listati di programmi assembler; in essi:

- la prima colonna riporta un numero in esadecimale, preceduto da virgola, che è l'indirizzo del primo byte dell'istruzione scritta nella riga;
- le tre colonne seguenti (che possono ridursi solo a una o due) rappresentano in codice macchina, byte a byte, l'istruzione in esadecimale;
- le successive colonne rappresentano l'istruzione in linguaggio simbolico assembler (dove $ rappresenta un numero esadecimale);
- i commenti iniziano sempre con un punto e virgola.

GRAF12

```
,C000 A9 20     LDA #$20
  ;20 = Prima Pagina della Pagina grafica
,C002 85 FF     STA $FF
  ;Piu' sign. Puntatore in Pagina zero
,C004 A9 00     LDA #$00
,C006 85 FE     STA $FE
  ;meno sign. Puntatore in Pagina zero
```

```
,C008 A9 00     LDA #$00
,C00A A8        TAY
,C00B 91 FE     STA ($FE),Y
  ;Pulisce la cella Ponendovi 0
,C00D C8        INY
,C00E D0 FB     BNE $C00B
  ;se non e' finita la Pagina continua
,C010 E6 FF     INC $FF
  ;incrementa il Puntatore
,C012 A5 FF     LDA $FF
,C014 C9 40     CMP #$40
  ;40 = ult. Pag. della Pagina grafica +1
,,C016 D0 F0    BNE $C008
  ;se non ha finito continua
,C018 AD 20 CB LDA $CB20
  ;52000 decimale:
  ;contiene il colore desiderato
,C01B 99 00 04 STA $0400,Y
  ;Pone nell' y-mo byte della
  ;Prima Pagina mappa video
,C01E 99 00 05 STA $0500,Y
  ;Pone nell' y-mo byte della
  ;seconda Pagina mappa video
,C021 99 00 06 STA $0600,Y
  ;Pone nell' y-mo byte della
  ;terza Pagina mappa video
,C024 99 00 07 STA $0700,Y
  ;Pone nell' y-mo byte della
  ;quarta Pagina mappa video
,C027 C8        INY
,C028 D0 F1     BNE $C01B
  ;se fine Pagina esce
,C02A 60        RTS
```

Per caricare GRAF12 nei tuoi programmi BASIC basta scrivere all'inizio del programma le istruzioni:

```
FOR I = 49152 TO 49194
READ A
POKE I,A: NEXT I
```

e scrivere le linee DATA riportate in GRAF13.

```
1 REM GRAF13
1000 DATA169,32,133,255,169,0,133,254
1010 DATA169,0,168,145,254,200,208,251
1020 DATA230,255,165,255,201,64,208,240
1030 DATA173,32,203,153,0,4,153,0
1040 DATA5,153,0,6,153,0,7,200
1050 DATA208,241,96
```

Ogni volta che vorrai usare la routine GRAF12, ti basterà inserire le linee sopra riportate, le linee DATA di GRAF13, porre in 52000 il valore desiderato del colore e dare il comando SYS 49152. Se vuoi solamente cambiare il colore dovrai dare il comando SYS 49186 (49186 (C018H)), dopo aver posto in 52000 il colore desiderato. Nota che la nostra routine pone in tutti i byte lo stesso colore anche se, per applicazioni più VARIOPINTE di quelle che illustreremo, è possibile dare a ogni matrice 8*8 un colore specifico.
Se la tua pagina grafica non parte dall'indirizzo 2000H, basta porre al posto del 32 (20H), che compare come secondo dato del programma, il numero corretto, e al posto del 64 (40H), che compare come 22-esimo dato del programma, il numero corretto, per esempio, se vuoi far partire la pagina grafica da 6000H, devi porre nella linea DATA 1000 96 (60H) al posto di 32 (secondo dato), e nella linea DATA 1020 128 (80H) al posto di 64 (sesto dato).
Ora che sappiamo come cancellare la pagina grafica, il problema che ci poniamo è come scriverci. Come fare cioè a DISEGNARE un solo punto tra i 64000 che formano la pagina? Cominciamo col dare un NOME ad ogni punto o, per meglio dire, due coordinate: X e Y. Chiamiamo il punto in basso a sinistra X=0,Y=0 e quello in alto a destra X=319,Y=199. Orientiamo cioè l'asse delle Y dal basso verso l'alto e l'asse delle X da sinistra verso destra. Consideriamo ora come ci appare sul video l'immagine degli 8000 byte che formano la pagina grafica: se chiamiamo OG l'indirizzo del primo byte della pagina grafica abbiamo lo schema riportato in Tabella 3.2.

```
OG+0*320+0*8+0   OG+0*320+1*8+0   ..  OG+0*320+39*8+0
OG+0*320+0*8+1   OG+0*320+1*8+1   ..  OG+0*320+39*8+1
OG+0*320+0*8+2   OG+0*320+1*8+2   ..  OG+0*320+39*8+2
OG+0*320+0*8+3   OG+0*320+1*8+3   ..  OG+0*320+39*8+3
OG+0*320+0*8+4   OG+0*320+1*8+4   ..  OG+0*320+39*8+4
```

OG+0*320+0*8+5	OG+0*320+1*8+5	..	OG+0*320+39*8+5
OG+0*320+0*8+6	OG+0*320+1*8+6	..	OG+0*320+39*8+6
OG+0*320+0*8+7	OG+0*320+1*8+7	..	OG+0*320+39*8+7
OG+1*320+0*8+0	OG+1*320+1*8+0	..	OG+1*320+39*8+0
OG+1*320+0*8+1	OG+1*320+1*8+1	..	OG+1*320+39*8+1
OG+1*320+0*8+2	OG+1*320+1*8+2	..	OG+1*320+39*8+2
OG+1*320+0*8+3	OG+1*320+1*8+3	..	OG+1*320+39*8+3
OG+1*320+0*8+4	OG+1*320+1*8+4	..	OG+1*320+39*8+4
OG+1*320+0*8+5	OG+1*320+1*8+5	..	OG+1*320+39*8+5
OG+1*320+0*8+6	OG+1*320+1*8+6	..	OG+1*320+39*8+6
OG+1*320+0*8+7	OG+1*320+1*8+7	..	OG+1*320+39*8+7
.	.	.	.
.	.	.	.
OG+24*320+0*8+0	OG+24*320+1*8+0	..	OG+24*320+39*8+0
OG+24*320+0*8+1	OG+24*320+1*8+1	..	OG+24*320+39*8+1
OG+24*320+0*8+2	OG+24*320+1*8+2	..	OG+24*320+39*8+2
OG+24*320+0*8+3	OG+24*320+1*8+3	..	OG+24*320+39*8+3
OG+24*320+0*8+4	OG+24*320+1*8+4	..	OG+24*320+39*8+4
OG+24*320+0*8+5	OG+24*320+1*8+5	..	OG+24*320+39*8+5
OG+24*320+0*8+6	OG+24*320+1*8+6	..	OG+24*320+39*8+6
OG+24*320+0*8+7	OG+24*320+1*8+7	..	OG+24*320+39*8+7

Tabella 3.2 Indirizzi dei byte nella pagina grafica

Come puoi vedere dalla Tabella 3.2, abbiamo per ogni byte un indirizzo del tipo:
OG+A*320+B*8+C
dove A indica la linea di caratteri contando dall'alto verso il basso (da 0 a 24) che contiene il byte desiderato, B indica la posizione del carattere nella linea (da 0 a 39) che contiene il byte e C è il numero del byte del carattere (da 0 a 7). Se vogliamo esprimere A, B e C in funzione di X e Y abbiamo:

```
A=INT((199-Y)/8)
B=INT(X/8)
C=199-Y-A*8
```

Cerchiamo di capire le relazioni precedenti. Poichè numeriamo le linee da 0 a 199, 199 è il più alto numero di linea. Sottraendo a 199 il valore della Y troviamo in quale linea di punti, contando dall'alto, si trova il punto che vogliamo disegnare; in altre parole abbiamo ribaltato l'asse delle Y. Se ora dividiamo per 8 il numero così ottenuto abbiamo, come quoziente A e come resto C. Dividendo la X per 8, si ottiene come quoziente la colonna di caratteri che contiene il byte che ci interessa cioè B; il resto di questa divisione ci dice, invece, qual'è il bit (pixel) che ci interessa. A questo punto sappiamo come annerire un punto X,Y dello schermo: basta dare al COMMODORE 64 i seguenti ordini:

```
A=INT((199-Y)/8)
B=INT(X/8)
C=199-Y-A*8
D=X-B
BY=OG+A*320+B*8+C
POKE A,PEEK(A) OR 2↑(7-D)
```

La variabile D contiene il resto della divisione di X per 8, e cioè il bit, contando da sinistra, che ci interessa. La variabile OG deve contenere già l'indirizzo del primo byte della pagina grafica. L'ultimo comando è quello che effettivamente DISEGNA il punto; pone infatti nel byte interessato il valore che ci interessa: 2 elevato a (7-D) è il numero binario formato da tutti zeri tranne il D-esimo bit a partire da sinistra. Nel byte viene posto il risultato della OR tra questo numero e il contenuto precedente del byte, per non CANCELLARE altri eventuali punti già disegnati in quel byte (vedi Paragrafo 1.2).
Purtroppo, molto spesso, in un'applicazione grafica i punti da disegnare sono moltissimi e quindi sarebbe meglio poter rendere più veloce la routine che disegna un punto sul video. Ancora una volta la soluzione è una routine in assembler, la GRAF14.

GRAF14

```
,C02B A9 C8     LDA #$C8
  ;C8 = 200 dec. = numero di punti
  ;in verticale
,C02D 38        SEC
,C02E ED 20 CB SBC $CB20
  ;sottrae la Y voluta
  ;(l'asse delle Y e' verso l'alto)
```

```
,C031 8D 20 CB STA $CB20
  ;Pone il risultato in Y
,C034 29 F8    AND #$F8
  ;azzera i 3 bit meno significativi
,C036 8D 1F CB STA $CB1F
  ;Pone il risultato in LN (linea)
,C039 AD 21 CB LDA $CB21
  ;byte meno significativo della X voluta
,C03C 29 F8    AND #$F8
  ;azzera i 3 bit meno significativi
,C03E 8D 1E CB STA $CB1E
  ;Pone il risultato in CL (colonna)
,C041 AD 20 CB LDA $CB20      ;Y
,C044 29 07    AND #$07
  ;azzera i 5 bit Piu' significativi
,C046 85 FE    STA $FE
  ;meno significativo
  ;Puntatore in Pagina zero
,C048 AD 21 CB LDA $CB21
  ;meno significativo X
,C04B 29 07    AND #$07
  ;azzera i 5 bit Piu' significativi
,C04D 8D 1D CB STA $CB1D
  ;Pone il risultato in BIT
  ;(bit che sara' "disegnato"
  ;Partendo da sinistra)
,C050 A5 FE    LDA $FE
  ;numero di byte in seno alla matrice 8*8
  ;che deve essere modificato
,C052 18       CLC
,C053 6D 1E CB ADC $CB1E
  ;aggiunge CL
,C056 85 FE    STA $FE
  ;Pone il risultato nel Puntatore
,C058 A9 20    LDA #$20
  ;Prima Pagina della Pagina grafica
,C05A 6D 22 CB ADC $CB22
  ;aggiunge Piu' significativo della X
,C05D 85 FF    STA $FF
  ;Pone il risultato nel byte
  ;Piu' significf. del Puntatore
,C05F A9 00    LDA #$00
```

```
,C061 8D 1C CB STA $CB1C
  ;azzera un byte per memoria temporanea
,C064 AD 1F CB LDA $CB1F
  ;NL
,C067 18       CLC
,C068 2A       ROL
,C069 8D 1B CB STA $CB1B
,C06C 2E 1C CB ROL $CB1C
,C06F 2E 1B CB ROL $CB1B
,C072 2E 1C CB ROL $CB1C
  ;CB1B-CB1C contiene ora NL*4
,C075 AD 1F CB LDA $CB1F
,C078 6D 1B CB ADC $CB1B
,C07B 8D 1B CB STA $CB1B
,C07E A9 00    LDA #$00
,C080 6D 1C CB ADC $CB1C
,C083 8D 1C CB STA $CB1C
  ;CB1B-CB1C contiene ora NL*5
,C086 18       CLC
,C087 2E 1B CB ROL $CB1B
,C08A 2E 1C CB ROL $CB1C
,C08D 2E 1B CB ROL $CB1B
,C090 2E 1C CB ROL $CB1C
,C093 2E 1B CB ROL $CB1B
,C096 2E 1C CB ROL $CB1C
  ;CB1B-CB1C contiene ora NL*40
,C099 18       CLC
,C09A AD 1B CB LDA $CB1B
,C09D 65 FE    ADC $FE
,C09F 85 FE    STA $FE
,C0A1 AD 1C CB LDA $CB1C
,C0A4 65 FF    ADC $FF
,C0A6 85 FF    STA $FF
  ;il puntatore contiene l'indirizzo
  ;del byte da modificare
,C0A8 A9 80    LDA #$80
  ;bit a sinistra a 1, gli altri a 0
,C0AA AE 1D CB LDX $CB1D
  ;BIT
,C0AD F0 04    BEQ $C0B3
  ;se non bisogna shiftare
  ;il contenuto di A salta
```

```
,C0AF 4A        LSR
,C0B0 CA        DEX
,C0B1 D0 FC     BNE $C0AF
  ;shifta a destra BIT volte
,C0B3 A0 00     LDY #$00
,C0B5 11 FE     ORA ($FE),Y
,C0B7 91 FE     STA ($FE),Y
  ;disegna il Punto
,C0B8 60        RTS
```

Abbiamo posto l'inizio di GRAF14 in 49195 (C02BH) per poterlo tenere in memoria assieme al programma di pulizia della pagina grafica precedentemente illustrato. Anche questo programma, comunque è facilmente trasferibile poichè sono sempre stati usati salti con indirizzi relativi. Per caricare questa routine nei tuoi programmi BASIC ti basta scrivere all'inzio del programma le istruzioni:

```
FOR I = 49195 TO 49337
READ A
POKE I,A: NEXT I
```

e inserire le linee DATA riportate nel programma GRAF15.

```
1 REM GRAF15
1000 DATA169,200,56,237,32,203,141,32
1010 DATA203,41,248,141,31,203,173,33
1020 DATA203,41,248,141,30,203,173,32
1030 DATA203,41,7,133,254,173,33,203
1040 DATA41,7,141,29,203,165,254,24
1050 DATA109,30,203,133,254,169,32,109
1060 DATA34,203,133,255,169,0,141,28
1070 DATA203,173,31,203,24,42,141,27
1080 DATA203,46,28,203,46,27,203,46
1090 DATA28,203,173,31,203,109,27,203
1100 DATA141,27,203,169,0,109,28,203
1110 DATA141,28,203,24,46,27,203,46
1120 DATA28,203,46,27,203,46,28,203
1130 DATA46,27,203,46,28,203,24,173
1140 DATA27,203,101,254,133,254,173,28
```

```
1150 DATA203,101,255,133,255,169,128,174
1160 DATA29,203,240,4,74,202,208,252
1170 DATA160,0,17,254,145,254,96
```

Ogni volta che vorrai usare la routine GRAF14, ti basterà aggiungere le linee sopra riportate, le linee DATA di GRAF15, porre in 52000 il valore della Y, in 52001 il valore della X, parte meno significativa (cioè X AND 255), in 52002 il valore della X, parte più significativa (cioè X/256) e dare il comando SYS 49195.
Se la tua pagina grafica non parte dall'indirizzo 2000H, basta porre al posto del 32 (20H) che compare come quarantasettesimo dato del programma (il penultimo della sesta linea), il numero corretto (per esempio, se la pagina grafica parte da 6000H devi porre 96 (60H) al posto di 32).
Ora che abbiamo i mezzi per disegnare e cancellare abbastanza velocemente, proviamo un'applicazione con il programma GRAF16.

```
1 REM GRAF16
10 POKE53280,0:POKE53281,0
15 PRINTCHR$(147)CHR$(154)CHR$(14)CHR$(8)
20 POKE56,32:POKE55,0:CLR
30 FORI=0TO185:READA:POKE49152+I,A:NEXT
110 POKE52000,16:SYS49152
120 POKE53272,PEEK(53272)OR8
125 POKE53265,PEEK(53265)OR32
130 FORX=0TO319:I=14*π*(X/319)-7*π
135 Y=SIN(I)*I:Y=Y*90/(7*π)+100
140 GOSUB1000:NEXTX
150 GOTO150
1000 POKE52000,Y:POKE52001,XAND255
1010 POKE52002,X/256:SYS49195:RETURN
2000 DATA169,32,133,255,169,0,133,254
2010 DATA169,0,168,145,254,200,208,251
2020 DATA230,255,165,255,201,64,208,240
2030 DATA173,32,203,153,0,4,153,0
2040 DATA5,153,0,6,153,0,7,200
2050 DATA208,241,96,169,200,56,237,32
2060 DATA203,141,32,203,41,248,141,31
2070 DATA203,173,33,203,41,248,141,30
2080 DATA203,173,32,203,41,7,133,254
2090 DATA173,33,203,41,7,141,29,203
```

```
2100 DATA165,254,24,109,30,203,133,254
2110 DATA169,32,109,34,203,133,255,169
2120 DATA0,141,28,203,173,31,203,24
2130 DATA42,141,27,203,46,28,203,46
2140 DATA27,203,46,28,203,173,31,203
2150 DATA109,27,203,141,27,203,169,0
2160 DATA109,28,203,141,28,203,24,46
2170 DATA27,203,46,28,203,46,27,203
2180 DATA46,28,203,46,27,203,46,28
2190 DATA203,24,173,27,203,101,254,133
2200 DATA254,173,28,203,101,255,133,255
2210 DATA169,128,174,29,203,240,4,74
2220 DATA202,208,252,160,0,17,254,145
2230 DATA254,96
```

COMMENTO A GRAF16.
Linea 10: inizializza il video.
Linee 15-20: fine memoria a 2000H dove inizierà la pagina grafica.
Linea 30: mette in memoria, partendo dall'indirizzo 49152 (C000H), i programmi GRAF12 e GRAF14.
Linea 110: usando la routine GRAF12 pulisce la pagina grafica e pone come colore bianco su nero (bianco = 1, nero = 0 quindi 1*16+0=16 (bianco su nero)). Purtroppo, se il tuo monitor non è ottimo, data la grandezza dei punti, avrai degli strani effetti sul colore e il bianco risulterà di colori diversi in zone diverse.
Linee 120-125: entra in modo pagina grafica e pone la pagina in 8192 (2000H).
Linee 130-135: per tutte le 320 colonne, calcola il valore da dare alla Y per ottenere il grafico della funzione SIN(X)*X tra -7pigreca e 7pigreca.
Linea 140: salta alla routine 1000 e chiude il ciclo aperto nella linea 130.
Linea 150: ferma il programma. Premendo STOP/RESTORE il calcolatore torna alla normalità.
Linea 1000: routine che prepara i dati per la routine GRAF14 e la chiama. Verrà così disegnato un punto bianco in posizione X,Y.
Linee 2000-2230: contengono le linee DATA con i dati relativi alle due routine GRAF12 e GRAF14.

Come avrai notato la velocità con cui il calcolatore disegna la funzione non è delle più elevate: ciò è dovuto principalmente alla lentezza con cui il BASIC calcola il valore delle funzioni nelle linee 130-135. Se provi infatti a sostituirle con le seguenti linee:

```
130 FOR X = 0 TO 319
135 Y=X/2
```

il programma verrà svolto in metà tempo.
Il programma GRAF16 può ovviamente disegnare tutte le funzioni (che non abbiano però asintoti verticali) a patto che le linee 130-135 vengano sostituite da altre appropriate linee. Sarà comunque necessario fare ogni volta conti abbastanza noiosi per poter sfruttare tutto lo schermo e non uscirne. Poichè abbiamo un calcolatore, sarà il caso di far lavorare lui, e perciò abbiamo preparato il programma GRAF17.

```
1 REM GRAF17
5 REM SYNTAX ERROR SE FUNZIONE MAL DEFINITA
10 POKE53280,0:POKE53281,0
15 PRINTCHR$(147)CHR$(154)CHR$(14)CHR$(8)
20 POKE56,32:POKE55,0:CLR
30 FORI=0TO185:READA:POKE49152+I,A:NEXT
100 INPUT"F(X) ";F$
110 PRINTCHR$(147)CHR$(17)CHR$(17)CHR$(17);
115 PRINTCHR$(144)"140 DEFFNY(X)="F$
120 PRINT"GOTO 140"CHR$(19)
130 POKE198,2:POKE631,13:POKE632,13:END
140 DEFFNY(X)=SIN(X)
150 PRINTCHR$(154)CHR$(147)
160 INPUT"DOMINIO DA ";ID
170 INPUT"A ";FD
180 PRINT:PRINT"DISEGNO L'ASSE DELLE X ?"
190 GETA$:IFA$<>"S"ANDA$<>"N"THEN190
200 AX=1
210 IFA$="N"THENAX=0
220 PRINT:PRINT"CANCELLO LA MAPPA ?"
230 GETA$:IFA$<>"S"ANDA$<>"N"THEN230
240 CL=1
250 IFA$="N"THENCL=0
260 PRINT
265 PRINT"STO CALCOLANDO MASSIMO E MINIMO"
270 AD=FD-ID
280 FORX=0TO319
290 I=ID+X*AD/320
300 Y=FNY(I)
310 IFIR>YTHENIR=Y
320 IFFR<YTHENFR=Y
330 NEXT
```

```
340 AR=FR-IR
350 POKE52000,16:IFCLTHENSYS49152
355 SYS49176
360 POKE53272,PEEK(53272)OR8
370 POKE53265,PEEK(53265)OR32
380 ZR=ABS(IR/AR*199)
390 FORX=0TO319
400 I=ID+X*AD/320
410 Y=FNY(I)*199/AR+ZR
420 GOSUB1000
430 IFAXTHENY=ZR:GOSUB1000
440 NEXT
450 POKE198,0
460 GETA$:IFA$=""THEN460
470 POKE53272,PEEK(53272)AND247
475 POKE53265,PEEK(53265)AND223
480 PRINTCHR$(147)"VUOI UN'ALTRO GRAFICO ?"
490 GETA$:IFA$<>"S"ANDA$<>"N"THEN490
500 IFA$="N"THENSYS58260
510 RUN
1000 POKE52000,Y:POKE52001,XAND255
1010 POKE52002,X/256:SYS49195:RETURN
2000 DATA169,32,133,255,169,0,133,254
2010 DATA169,0,168,145,254,200,208,251
2020 DATA230,255,165,255,201,64,208,240
2030 DATA173,32,203,153,0,4,153,0
2040 DATA5,153,0,6,153,0,7,200
2050 DATA208,241,96,169,200,56,237,32
2060 DATA203,141,32,203,41,248,141,31
2070 DATA203,173,33,203,41,248,141,30
2080 DATA203,173,32,203,41,7,133,254
2090 DATA173,33,203,41,7,141,29,203
2100 DATA165,254,24,109,30,203,133,254
2110 DATA169,32,109,34,203,133,255,169
2120 DATA0,141,28,203,173,31,203,24
2130 DATA42,141,27,203,46,28,203,46
2140 DATA27,203,46,28,203,173,31,203
2150 DATA109,27,203,141,27,203,169,0
2160 DATA109,28,203,141,28,203,24,46
2170 DATA27,203,46,28,203,46,27,203
2180 DATA46,28,203,46,27,203,46,28
```

```
2190 DATA203,24,173,27,203,101,254,133
2200 DATA254,173,28,203,101,255,133,255
2210 DATA169,128,174,29,203,240,4,74
2220 DATA202,208,252,160,0,17,254,145
2230 DATA254,96
```

COMMENTO A GRAF17.
Linee 10-15: inizializza il video.
Linea 20: fine memoria a 2000H dove inizierà la pagina grafica.
Linea 30: mette in memoria, partendo dall'indirizzo 49152 (C000H), i programmi GRAF12 e GRAF14.
Linea 100: richiede la funzione che deve essere visualizzata. Questa viene posta nella variabile F$. Deve essere scritta secondo la normale sintassi del COMMODORE 64.
Linee 110-115: pulisce lo schermo e scrive nella quarta riga, in nero, DEFFNY(X)= seguita dalla definizione della funzione richiesta.
Linea 120: scrive sulla linea seguente il comando GOTO 140 e posiziona quindi il cursore nella posizione in alto a sinistra dello schermo.
Linea 130: scrive nel buffer della tastiera 2 RETURN (vedi il commento al programma GRAF6 del paragrafo 2.1.2) e esce. Il calcolatore scriverà quindi "READY" e si posizionerà sulla quarta linea dove: il primo RETURN inserirà nel programma la linea 140, e il secondo farà ricominciare l'esecuzione dalla linea 140.
Linea 150: pulisce nuovamente lo schermo e riporta il colore del cursore a blu chiaro.
Linee 160-170: chiede inizio e fine dell' intervallo in cui si vuole visualizzare la funzione.
Linee 180-210: se si vuole visualizzare l'asse delle ascisse la variabile AX viene posta a 1, altrimenti a 0.
Linee 220-250: se si vuole cancellare il vecchio grafico la variabile CL viene posta a 1, altrimenti a 0.
Linee 260-265: scrive il messaggio.
Linea 270: calcola l'ampiezza del dominio.
Linea 280: inizializza un ciclo.
Linee 290-300: calcola il valore della funzione su 320 punti equidistanti del dominio.
Linea 310: se la variabile IR (minimo) è maggiore del valore della funzione in questo punto, allora IR=F(I).
Linea 320: se la variabile FR (massimo) è minore del valore della funzione in questo punto, allora FR=F(I).
Linea 330: chiude il ciclo aperto nella linea 280.
Linea 340: calcola la differenza tra massimo e minimo della funzione.

Linea 350: se richiesto (CL=1) pulisce la pagina grafica e sceglie il colore bianco su nero.
Linee 360-370: entra in modo pagina grafica.
Linea 380: calcola il valore della linea di pixel che corrisponde all' asse delle ascisse.
Linee 390-440: con lo stesso procedimento con cui viene calcolato massimo e minimo, viene disegnata la funzione. Se richiesto (AX=1) nella linea 430 viene disegnato l'asse delle ascisse.
Linea 450: cancella i caratteri eventualmente contenuti nel buffer della tastiera.
Linea 460: attende che sia premuto un tasto.
Linee 470-475: ritorna al modo caratteri.
Linee 480-510: se richiesto riparte il programma, altrimenti salta alla routine di inizializzazione del BASIC.
Linea 1000: routine che prepara i dati per la routine GRAF14 e la chiama. Verrà così disegnato un punto bianco in posizione X,Y.
Linee 2000-2230: contengono i dati relativi alle due routine GRAF12 e GRAF14.

Se si presta un pò di attenzione è possibile disegnare anche funzioni con asintoti verticali: basta infatti fare in modo che il COMMODORE 64 non debba calcolare il valore della funzione nel punto in cui questa non è definita. Ad esempio volendo avere il grafico della funzione Y=1/X in un intorno dello 0 dovrai farla disegnare su un dominio che va, ad esempio, da -1 a 1.1, in modo che il valore più vicino allo 0, per cui il calcolatore calcolerà la funzione, sarà -2.49999994 E-3 per X = 153.
I risultati che si possono ottenere con GRAF17 sono già abbastanza simpatici ma il prossimo programma, GRAF18, intende dare risultati molto più spettacolari: disegna funzioni reali di due variabili reali, disegna cioè figure tridimensionali.

```
1 REM GRAF18
10 POKE56,32:POKE55,0:CLR
20 DIMX%(319),N%(319)
25 FORI=0TO319:N%(I)=200:NEXT
30 FORI=0TO185:READA:POKE49152+I,A:NEXT
40 POKE53280,0:POKE53281,0
50 POKE53272,PEEK(53272)OR8
60 POKE53265,PEEK(53265)OR32
70 POKE52000,16:SYS49152
80 X3=-2:FORY3=-2TO2STEP.02:GOSUB240:NEXT
90 SP=.3
100 FORX3=-2TO2STEP.02:Y3=-2
105 GOSUB240:Y3=2:GOSUB240
110 RX=X3-INT(X3/SP)*SP
```

```
120 FORY3=-2+RXTO2STEPSP
130 GOSUB240
140 NEXTY3
150 FORY3=2-RXTO-2STEP-SP
160 GOSUB240
170 NEXTY3:NEXTX3
180 X3=2:FORY3=-2TO2STEP.02:GOSUB240:NEXT
190 POKE198,0
200 GETA$:IFA$=""THEN200
210 POKE53272,PEEK(53272)AND247
220 POKE53265,PEEK(53265)AND223
230 PRINTCHR$(147):END
240 Z3=Y3*Y3/4-X3*X3/4
245 X2=160+(Y3+X3/2)*49:Y2=100+(Z3+X3/2)*49
250 IFY2>X%(X2)THENX%(X2)=Y2:GOTO280
260 IFY2<N%(X2)THENN%(X2)=Y2:GOTO290
270 RETURN
280 IFY2<N%(X2)THENN%(X2)=Y2
290 GOSUB300:RETURN
300 POKE52000,Y2:POKE52001,X2AND255
310 POKE52002,X2/256:SYS49195:RETURN
1000 DATA169,32,133,255,169,0,133,254
1010 DATA169,0,168,145,254,200,208,251
1020 DATA230,255,165,255,201,64,208,240
1030 DATA173,32,203,153,0,4,153,0
1040 DATA5,153,0,6,153,0,7,200
1050 DATA208,241,96,169,200,56,237,32
1060 DATA203,141,32,203,41,248,141,31
1070 DATA203,173,33,203,41,248,141,30
1080 DATA203,173,32,203,41,7,133,254
1090 DATA173,33,203,41,7,141,29,203
1100 DATA165,254,24,109,30,203,133,254
1110 DATA169,32,109,34,203,133,255,169
1120 DATA0,141,28,203,173,31,203,24
1130 DATA42,141,27,203,46,28,203,46
1140 DATA27,203,46,28,203,173,31,203
1150 DATA109,27,203,141,27,203,169,0
1160 DATA109,28,203,141,28,203,24,46
1170 DATA27,203,46,28,203,46,27,203
1180 DATA46,28,203,46,27,203,46,28
1190 DATA203,24,173,27,203,101,254,133
```

```
1200 DATA254,173,28,203,101,255,133,255
1210 DATA169,128,174,29,203,240,4,74
1220 DATA202,208,252,160,0,17,254,145
1230 DATA254,96
```

COMMENTO A GRAF18.
Linea 10: pone la fine della memoria a 8192 (2000H).
Linea 20: dimensiona i vettori X% (max) e N% (min) di 320 elementi, quante sono le colonne di punti nel video. Questi serviranno a sapere se il punto che si stà per disegnare è coperto dalla funzione o no. Dopo aver dimensionato i vettori pone in tutti gli elementi di N% il numero 200.
Linea 30: carica in memoria le routine GRAF12 e GRAF14.
Linea 40: schermo e sfondo neri.
Linea 50: entra nel modo pagina grafica.
Linea 60: pone la pagina grafica in 8192 (2000H).
Linea 70: sceglie bianco su nero e salta alla routine GRAF12.
Linea 80: calcola e disegna in assonometria (appoggiandosi alla routine in 240) il valore della funzione sul segmento X=-2, -2 < Y < 2, cioè il segmento più vicino all'osservatore. I punti del segmento su cui viene fatto il conto sono molto vicini (passo=0.02 cioè 1/50 del segmento stesso).
Linea 90: pone il passo del reticolo=0.3. Aumentando il valore della variabile SP si otterrà un reticolo più largo, diminuendolo un reticolo più stretto.
Linee 100-105: inizializza un ciclo che incrementa la X (asse che va dall'osservatore verso lo schermo) con passo di 0.02. Quindi calcola e disegna il valore della funzione per Y=-2 e per Y=2 cioè ai due lati del dominio.
Linee 110-120: inizializza un ciclo che incrementa la Y con passo SP (in questo caso 0.3) partendo da un valore di Y compreso tra -2-SP e -2+SP in modo che il reticolo sia diagonale rispetto al dominio.
Linea 130: calcola e disegna il valore della funzione per quei valori di X e Y.
Linea 140: chiude il ciclo della Y.
Linee 150-170: esegue lo stesso lavoro per i valori della Y simmetrici per disegnare le diagonali perpendicolari alle prime.
Linea 170: chiude anche il ciclo della X.
Linea 180: come la linea 80 solo che il segmento è X=2 (il più lontano dall'osservatore).
Linea 190: svuota il buffer di tastiera.
Linea 200: attende che sia premuto un tasto.
Linee 210-220: riportano il video in modo caratteri.
Linea 230: pulisce lo schermo e termina il programma.
Linee 240-245: calcola il valore della funzione tridimensionale (Z3) e quindi i valori delle X e Y bidimensionali cioè quelli dell'assonometria.

Linea 250: se il valore massimo della Y2 su quella colonna è minore del valore appena calcolato, allora pone il valore del massimo sulla colonna uguale al valore appena calcolato, e salta alla linea 280.
Linea 260: se il valore minimo della Y2 su quella colonna è maggiore del valore appena calcolato, allora pone il valore del minimo sulla colonna uguale al valore appena calcolato, e salta alla linea 290.
Linea 280: se il valore della Y2 cade tra massimo e minimo il punto è nascosto dalla funzione precedentemente disegnata e quindi non deve essere disegnato.
Linea 290: il programma arriva qui se il valore della Y è maggiore del massimo della Y su quella colonna. Qui, se il valore della Y è anche minore del minimo su questa colonna pone minimo uguale al valore appena calcolato. Questo caso si verifica quando si disegna il primo punto su una colonna: infatti all'inizializzazione: N%=200 e X%=0. Salta alla routine che disegna il punto e ritorna.
Linee 300-310: prepara la routine GRAF14 e la chiama, quindi ritorna.
Linee 1000-1230: contengono i dati relativi alle routine GRAF12 e GRAF14.

Come hai visto questo programma permette di disegnare una porzione di funzione tridimensionale e, per essere più precisi disegna una porzione di spazio $-2<X<2$, $-2<Y<2$, $-1<Z<1$. Se vuoi vedere una funzione in uno spazio maggiore o minore di questo conviene modificare la funzione stessa nelle linee 240-245. Ad esempio, se si vuole vedere la funzione F(X,Y) su un dominio $-10<X<10$, $-20<Y<20$, basterà impostare la funzione F(X*5,Y*10)/M, dove M è il massimo che ABS(F(X*5,Y*10)) assume nell'intervallo $-2<X<2$, $-2<Y<2$, o (che è la stessa cosa) che la funzione F(X,Y) assume nell'intervallo $-10<X<10$, $-20<Y<20$. In ogni caso la funzione sarà centrata nell'origine (X=0,Y=0). Se preferisci puoi modificare il programma, tenendo conto delle regole appena mostrate, approssimativamente come e stato fatto per il programma GRAF16: l'unico svantaggio che può comportare un'operazione di questo tipo è il tempo che impiegherà il programma per calcolare massimo e minimo.
Per ottenere figure diverse da quella che si ottiene normalmente, prova a far girare il programma sostituendo la linea 240 con una delle seguenti:

```
240 Z3=SIN(Y3*2)
```

oppure:

```
240 Z3=SIN(X3*X3+Y3*Y3)
```

oppure.

```
240 Z3=(X3*X3+Y3*Y3)*(X3*X3+Y3*Y3)/32-1
```

Riportiamo nelle Figure 3.1, 3.2, 3.3 e 3.4 i grafici ottenuti con il programma GRAF18, come appare nel listato, e con le 3 modifiche della linea 240 sopra indicate.

```
240 Z3=Y3*Y3/4-X3*X3/4
```

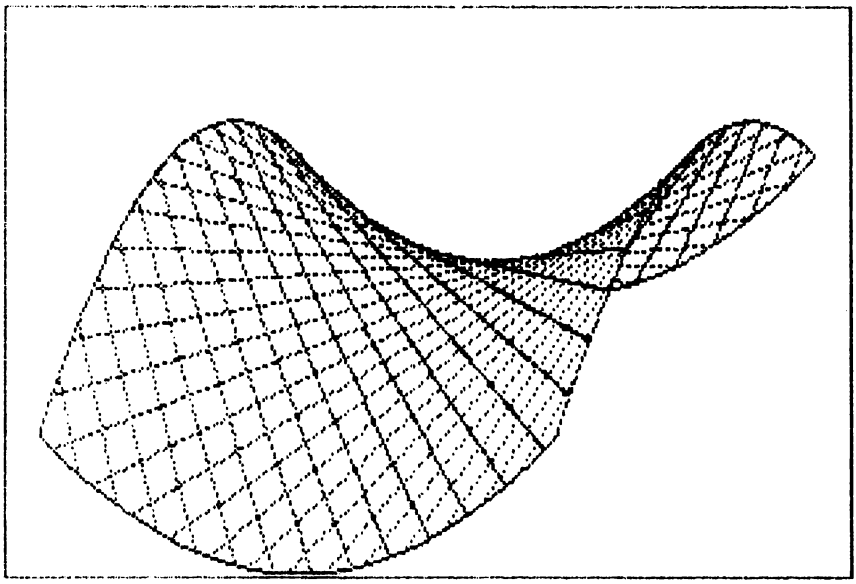


Figura 3.1 Grafico ottenuto con il programma GRAF18

```
240 Z3=SIN(Y3*2)
```

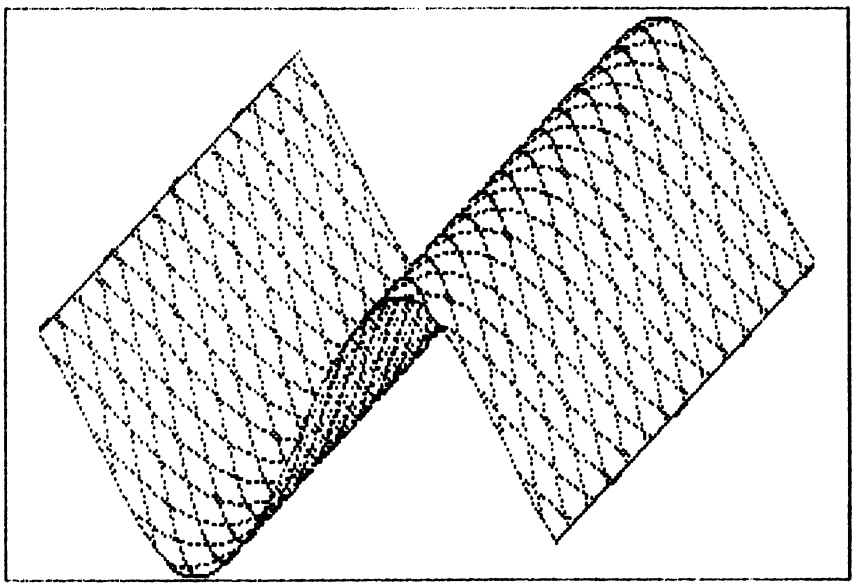


Figura 3.2 Grafico ottenuto con il programma GRAF18 con modifica 1

```
240 Z3=SIN(X3*X3+Y3*Y3)
```

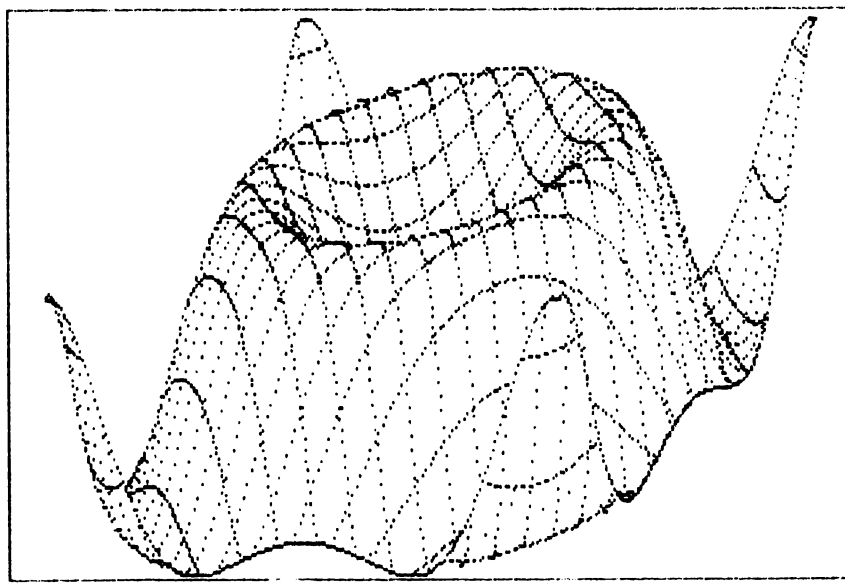


Figura 3.3 Grafico ottenuto con il programma GRAF18 con modifica 2

```
240 Z3=(X3*X3+Y3*Y3)*(X3*X3+Y3*Y3)/32-1
```

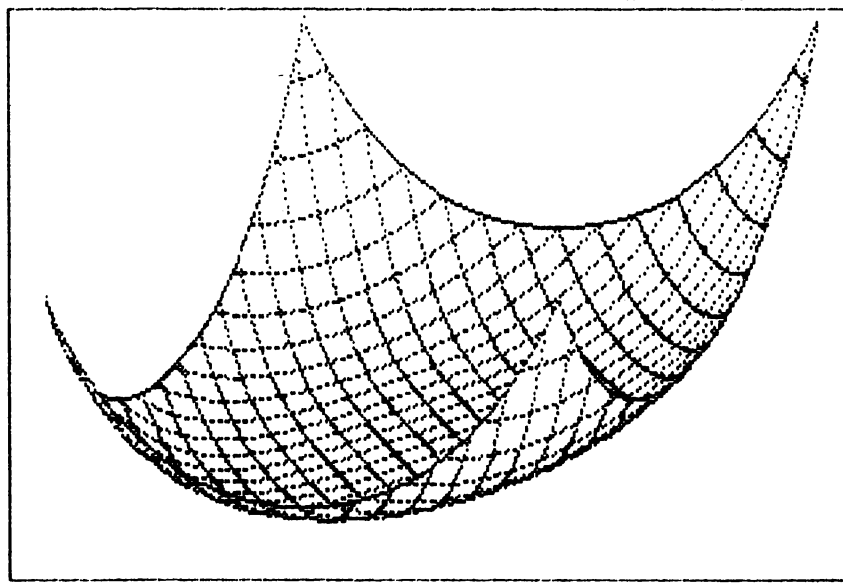


Figura 3.4 Grafico ottenuto con il programma GRAF18 con modifica 3

3.1.2 Pagina grafica multicolore

Come per i caratteri, anche per la pagina grafica nel modo multicolore ogni punto dello schermo corrisponde a 2 bit della memoria e la risoluzione orizzontale viene quindi dimezzata (i punti sono ora SOLO 32000 disposti su 200 linee di 160 punti). Come per i caratteri, anche per la pagina grafica, il comando da dare al COMMODORE 64 perchè entri in modo multicolore è:
POKE 53270,PEEK(53270) OR 16
bisogna cioè porre a 1 il bit di posizione 4 del registro 22 (16H) del VIC II. La differenza che esiste tra il modo multicolore con i caratteri e il mòdo multicolore con la pagina grafica è nella scelta dei colori: nel modo pagina grafica ogni matrice 4*8 può avere, indipendentemente dalle altre, 3 colori diversi scelti tra i 16 disponibili e un solo colore in comune con le altre, anch'esso scelto tra i 16 a disposizione. Il colore comune a tutte le matrici è quello dello schermo, mentre gli altri 3 vengono posti:
A) nei 4 bit più significativi della memoria video (che, come abbiamo già visto, viene utilizzata come mappa del colore, quando il COMMODORE 64 è in modo pagina grafica).
B) nei 4 bit meno significativi della memoria video.
C) nella mappa del colore.
Per la matrice 4*8 posta in alto a sinistra vale quindi quanto riportato nella Tabella 3.3.

BIT	COLORE DEL PUNTO CORRISPONDENTE
00	colore dello schermo
01	colore contenuto nei 4 bit più significativi del primo byte della memoria video
10	colore contenuto nei 4 bit meno significativi del primo byte della memoria video
11	colore contenuto nel primo byte della memoria del colore

Tabella 3.3 Corrispondenze tra COPPIE di BIT e COLORI in multicolore

Usando le routine che abbiamo già visto per la gestione della pagina grafica, si può facilmente gestire anche la pagina grafica multicolore: eccone un esempio nel programma GRAF19.

```
1 REM GRAF19
10 POKE56,32:POKE55,0:CLR
20 DIMX%(159),N%(159)
25 FORI=0TO159:N%(I)=200:NEXT
30 FORI=0TO185:READA:POKE49152+I,A:NEXT
40 POKE53280,0:POKE53281,0
50 POKE53272,PEEK(53272)OR8
55 POKE53270,PEEK(53270)OR16
60 POKE53265,PEEK(53265)OR32
70 POKE52000,39:SYS49152
80 X3=-2:FORY3=-2TO2STEP.02:GOSUB240:NEXT
90 SP=.3
100 FORX3=-2TO2STEP.02:Y3=-2
105 GOSUB240:Y3=2:GOSUB240
110 RX=X3-INT(X3/SP)*SP
120 FORY3=-2+RXTO2STEPSP
130 GOSUB240
140 NEXTY3
150 FORY3=2-RXTO-2STEP-SP
160 GOSUB240
170 NEXTY3:NEXTX3
180 X3=2:FORY3=-2TO2STEP.02:GOSUB240:NEXT
190 POKE198,0
200 GETA$:IFA$=""THEN200
210 POKE53272,PEEK(53272)AND247
215 POKE53270,PEEK(53270)AND239
220 POKE53265,PEEK(53265)AND223
230 PRINTCHR$(147):END
240 Z3=SIN(X3*3)
245 X2=INT(80+(Y3+X3/2)*24):Y2=100+(Z3+X3/2)*49
250 IFY2>X%(X2)THENX%(X2)=Y2:CL=0:GOTO280
260 IFY2<N%(X2)THENN%(X2)=Y2:CL=1:GOTO300
270 RETURN
280 IFY2<N%(X2)THENN%(X2)=Y2
290 GOSUB300:RETURN
300 IFCLTHENX=X2*2+1:GOSUB305:RETURN
302 X=X2*2
305 POKE52000,Y2:POKE52001,XAND255
310 POKE52002,X/256:SYS49195:RETURN
1000 DATA169,32,133,255,169,0,133,254
1010 DATA169,0,168,145,254,200,208,251
1020 DATA230,255,165,255,201,64,208,240
```

```
1030 DATA173,32,203,153,0,4,153,0
1040 DATA5,153,0,6,153,0,7,200
1050 DATA208,241,96,169,200,56,237,32
1060 DATA203,141,32,203,41,248,141,31
1070 DATA203,173,33,203,41,248,141,30
1080 DATA203,173,32,203,41,7,133,254
1090 DATA173,33,203,41,7,141,29,203
1100 DATA165,254,24,109,30,203,133,254
1110 DATA169,32,109,34,203,133,255,169
1120 DATA0,141,28,203,173,31,203,24
1130 DATA42,141,27,203,46,28,203,46
1140 DATA27,203,46,28,203,173,31,203
1150 DATA109,27,203,141,27,203,169,0
1160 DATA109,28,203,141,28,203,24,46
1170 DATA27,203,46,28,203,46,27,203
1180 DATA46,28,203,46,27,203,46,28
1190 DATA203,24,173,27,203,101,254,133
1200 DATA254,173,28,203,101,255,133,255
1210 DATA169,128,174,29,203,240,4,74
1220 DATA202,208,252,160,0,17,254,145
1230 DATA254,96
```

Il programma GRAF19 differisce dal precedente solo per pochissime cose che gli permettono di disegnare la funzione 3-D con la parte superiore di un colore e quella inferiore di un altro. Vediamo queste differenze nel commento.

COMMENTO A GRAF19.
Linea 20: i vettori N% e X% hanno solo 160 elementi poichè la risoluzione orizzontale è dimezzata.
Linea 55: entra anche in modo multicolore.
Linea 70: i colori scelti sono rosso e giallo.
Linee 240-245: abbiamo scelto una funzione diversa e il calcolo della X2 è diverso a causa della diversa risoluzione del video.
Linee 250-260: se il punto viene disegnato sopra, la variabile CL viene posta a 0 altrimenti a 1.
Linea 305: è la vecchia 300.
Linea 300: se CL=1 annerisce il bit meno significativo della coppia di bit che ci interessa.
Linea 302: se CL=0 annerisce il bit più significativo della coppia di bit che ci interessa.

Anche in questo caso puoi sostituire la linea 240 con una di quelle suggerite per modificare il programma GRAF18 e provare il programma.

CAPITOLO 4

ALTRE POSSIBILITA'

4.1 ALTRE POSSIBILITA' DEL VIC II

Vediamo ora le caratteristiche del VIC II che non abbiamo ancora considerato.

4.1.1. Annullamento dello schermo (Screen Blanking)

E' possibile annullare tutto ciò che viene visualizzato sullo schermo ponendo a zero il bit di posizione 4 del registro 17 (11H) del VIC II. Tale registro risponde all'indirizzo 53265 (D011H): il comando da dare al COMMODORE 64 è quindi:
POKE 53265,PEEK(53265) AND 239.
Compiendo questa operazione lo schermo diventa completamente vuoto e dello stesso colore del bordo (come quando il calcolatore carica da nastro). Per riportare la situazione alla normalità bisogna compiere l'operazione inversa: dare cioè il comando:
POKE 53265,PEEK(53265) OR 16.
Si può sfruttare questa opzione del VIC II per compiere delle operazioni sullo schermo mentre questo non si può vedere, ma il vero significato di questo bit è un altro.
Come sai, sia il VIC II che la CPU accedono alla memoria. Come fanno a non litigare mai? Cioè, come è possibile che non vogliano leggere entrambi, nello stesso momento, due byte di memoria diversi? Ciò sarebbe impossibile perchè la memoria accetta un solo indirizzo alla volta e può presentare o accettare un solo dato alla volta. Esiste un segnale importantissimo all'interno del calcolatore che temporizza tutto il sistema e prende il nome di CLOCK. Questo segnale non è altro che un'onda quadra di frequenza ben precisa (nel nostro caso 0.98 MHz); cioè il clock, ogni secondo, passa dallo stato ALTO allo stato BASSO (o viceversa) 980000 volte. La CPU, che ha un piedino che è collegato al clock, è stata costruita in modo che acceda alla memoria solamente quando il clock è alto. Al VIC II rimangono quindi, per leggere la memoria tutti i periodi in cui il clock è basso. Poichè il clock rimane basso per 1/1960000 di secondo (cioè per poco più di mezzo microsecondo), ogni volta che il clock è basso il VIC II ha tempo per fare una sola lettura: potrà quindi leggere un dato ogni microsecondo. Ma in alcuni casi il VIC II, per eseguire

correttamente le sue funzioni, deve leggere dati dalla memoria con una frequenza più elevata e deve quindi chiedere alla CPU di leggere la memoria anche quando il clock è alto, rubandole tempo. Il nostro famoso bit quindi regala tempo alla CPU visto che il VIC II, non dovendo più disegnare nè il contenuto del quadro nè gli sprite, non chiederà più tempo alla CPU quando non gli spetta. Prova a far girare il programma GRAF20.

```
1 REM GRAF20
10 POKE53265,PEEK(53265)AND239
20 TI$="000000"
30 FORI=1TO10000:NEXT
40 A=TI/60
50 POKE53265,PEEK(53265)OR16
60 TI$="000000"
70 FORI=1TO10000:NEXT
80 B=TI/60
90 PRINT"CON SCHERMO ANNULLATO :"A"SECONDI"
100 PRINT"CON SCHERMO NON ANNULLATO :"B"SECONDI"
```

Come vedi il tempo impiegato a schermo annullato è minore di quello impiegato a schermo pieno. Un altro, più importante vantaggio di questa opzione del VIC II consiste nelle routine di ritardo: in tali routine, se scritte in linguaggio macchina, il ritardo può essere calcolato al milionesimo di secondo annullando lo schermo e disabilitando gli interrupt. Questi ritardi cosi precisi servono nel caso di alcune operazioni di I/O come, ad esempio, quelle del nastro.

4.1.2 Il registro di linea e i registri di interrupt

Il registro di linea è un registro di 9 bit, i cui 8 meno significativi sono posti nel registro 18 del VIC II (indirizzo 53266 (D012H)) e il più significativo è il bit di posizione 7 del registro 17 (indirizzo 53265 (D011H)). Lo scopo di questo registro è duplice: quando viene letto dà la posizione della linea del pennello elettronico del televisore. Quando vi si scrive (compreso il bit più significativo) il numero viene memorizzato e quando la linea diventa uguale al numero scritto il VIC II lancia un INTERRUPT (IRQ).
Per sapere come gestire questo interrupt bisogna conoscere il significato di due registri molto importanti del VIC II: il registro di stato degli interrupt e il registro di abilitazione degli interrupt.

Il REGISTRO DI STATO DEGLI INTERRUPT rispode all'indirizzo 53273 (D019H) e svolge due funzioni come il registro di linea; quando viene letto segnala quale interrupt (di quelli che possono essere generati dal VIC II) è pendente.

La segnalazione avviene in questo modo:

- bit di posizione 0: se è a 1, è pendente l'interrupt generato dalla funzione di linea;
- bit di posizione 1: se è a 1, è pendente l'interrupt generato da una collisione tra sprite e dati;
- bit di posizione 2: se è a 1, è pendente l'interrupt generato da una collisione tra sprite;
- bit di posizione 3: se è a 1, è pendente l'interrupt generato da una transizione negativa della light pen;
- bit di posizione 7: è a 1 ogni volta che uno dei 4 interrupt è pendente.

Quando, invece, si scrive nel registro di stato degli interrupt, si avvisa il VIC II che si è pronti a ricevere un altro interrupt dello stesso tipo.
Ad esempio, se due sprite si scontrano, viene lanciato un interrupt alla CPU e viene posto a 1 il bit di posizione 2 del registro di stato degli interrupt. Quando la CPU è pronta ad accettare un altro interrupt scrive 1 nello stesso bit dello stesso registro, avvisando così il VIC II che può generare un altro interrupt.

Il REGISTRO DI ABILITAZIONE DEGLI INTERRUPT risponde all'indirizzo 53274 (D01AH) e ha lo stesso formato del registro di stato degli interrupt. Se un bit di questo registro è a 1, l'interrupt corrispondente è abilitato, se è a 0, quando si verifica una condizione tra quelle sopra esposte, non viene chiamato alcun interrupt, ma viene ugualmente posto a 1 il bit corrispondente del registro di stato degli interrupt.
Ovviamente, per poter usufruire di questa versatile struttura di interrupt bisogna programmare in linguaggio macchina.
Il programma GRAF21, che segue, è un esempio istruttivo di come possano essere usati i registri di linea e di interrupt.

GRAF21

```
,C000 78        SEI
  ;disabilita l'interrupt
,C001 A9 C0     LDA #$C0
,C003 8D 15 03  STA $0315
,C006 A9 26     LDA #$26
```

```
,C008 8D 14 03 STA $0314
  ;la nuova routine di interrupt
  ;parte ora da C026
,C00B A9 FA    LDA #$FA
,C00D 8D 12 D0 STA $D012
,C010 AD 11 D0 LDA $D011
,C013 29 7F    AND #$7F
,C015 8D 11 D0 STA $D011
  ;il VIC II lancera' un interrupt quando
  ;il raster conterra' $FA (fine schermo)
,C018 AD 1A D0 LDA $D01A
,C01B 09 01    ORA #$01
,C01D 8D 1A D0 STA $D01A
  ;abilita l'interrupt del raster
,C020 A9 FF    LDA #$FF
,C022 85 FF    STA $FF
  ;inizializza il flag che avvisa in che
  ;posizione dello schermo e' stato
  ;lanciato l'interrupt
,C024 58       CLI
  ;accetta gli interrupt
,C025 60       RTS
  ;torna al BASIC
,C026 AD 19 D0 LDA $D019
  ;NUOVA ROUTINE DI INTERRUPT
,C029 29 01    AND #$01
,C02B D0 03    BNE $C030
  ;se l'interrupt e' stato chiamato
  ;da VIC II salta
,C02D 4C 31 EA JMP $EA31
  ;altrimenti va alla normale routine
  ;di interrupt
,C030 A9 01    LDA #$01
,C032 8D 19 D0 STA $D019
  ;avvisa il VIC II che ha accettato
  ;l'interrupt
,C035 AD 20 D0 LDA $D020
,C038 49 01    EOR #$01
,C03A 8D 20 D0 STA $D020
  ;cambia colore al bordo
,C03D A5 FF    LDA $FF
,C03F D0 10    BNE $C051
```

```
  ;se non siamo all'inizio dello
  ;schermo salta
,C041 A9 FA     LDA #$FA
,C043 8D 12 D0 STA $D012
,C046 AD 11 D0 LDA $D011
,C049 29 7F     AND #$7F
,C04B 8D 11 D0 STA $D011
  ;chiede il prossimo interrupt
  ;alla linea $FA (fine schermo)
,C04E 18        CLC
,C04F 90 0D     BCC $C05E
  ;salta sempre al ritorno da interrupt
  ;(non abbiamo usato un'istruzione JMP
  ;per rendere il programma rilocabile)
,C051 A9 32     LDA #$32
,C053 8D 12 D0 STA $D012
,C056 AD 11 D0 LDA $D011
,C059 29 7F     AND #$7F
,C05B 8D 11 D0 STA $D011
  ;chiede il prossimo interrupt alla
  ;linea $32 (inizio schermo)
,C05E A9 FF     LDA #$FF
,C060 45 FF     EOR $FF
,C062 85 FF     STA $FF
  ;cambia il flag
,C064 68        PLA
,C065 A8        TAY
,C066 68        PLA
,C067 AA        TAX
,C068 68        PLA
,C069 40        RTI
  ;torna da interrupt
```

Per chi non avesse la possibilita di porre facilmente in memoria il programma in assembler presentiamo la versione realizzata con linee DATA, GRAF22.

```
1 REM GRAF22
10 FORI=49152TO49257:READA:POKEI,A:NEXT
20 SYS49152:NEW
1000 DATA120,169,192,141,21,3,169,38
1010 DATA141,20,3,169,250,141,18,208
1020 DATA173,17,208,41,127,141,17,208
1030 DATA173,26,208,9,1,141,26,208
1040 DATA169,255,133,255,88,96,173,25
1050 DATA208,41,1,208,3,76,49,234
1060 DATA169,1,141,25,208,173,32,208
1070 DATA73,1,141,32,208,165,255,208
1080 DATA16,169,250,141,18,208,173,17
1090 DATA208,41,127,141,17,208,24,144
1100 DATA13,169,50,141,18,208,173,17
1110 DATA208,41,127,141,17,208,169,255
1120 DATA69,255,133,255,104,168,104,170
1130 DATA104,64
```

Il risultato del programma GRAF22 è un quadro video un pò fuori dal comune, con la parte orizzontale del bordo di un colore e quella verticale di un altro colore. Questo effetto è stato ottenuto facendo in modo che il VIC II lanci un segnale di interrupt ogni volta che, nel disegnare il quadro video, il pennello elettronico è arrivato dove inizia la parte centrale dello schermo; a questo punto viene cambiato il colore del bordo. Riportiamo il colore del bordo al suo valore primitivo, con lo stesso procedimento, quando il pennello elettronico è arrivato alla fine della parte centrale dello schermo.

4.1.3 Scorrimento fine (Smooth Scrolling)

Il VIC II permette di far scorrere tutto lo schermo di un solo punto (un ottavo di carattere) sia in direzione verticale che orizzontale.
I registri con cui si controllano queste operazioni sono rispettivamente i registri 17 e 22 del VIC II che rispondono agli indirizzi 53265 (D011H) e 53270 (D016H). I tre bit meno significativi di questi registri indicano quale delle 8 possibili posizioni (da 0 a 7) debbano assumere i caratteri sul video, mentre il bit 3 di questi registri

seleziona, se posto a 0, rispettivamente il modo a 24 righe (normalmente sono 25) e a 38 colonne (normalmente sono 40).
Nota che lo schermo rimane sempre di 25 righe di 40 colonne, ma ponendo a zero quei bit vengono visualizzate solamente 24 righe e 38 colonne: questo permette di aggiungere, nella parte nascosta dello schermo, i dati nuovi e farli entrare lentamente nella parte visibile usando lo scorrimento fine. Per uno scorrimento fine devi:

1) diminuire la parte visibile dello schermo nella direzione in cui lo vuoi far scorrere (ad esempio, se vuoi far scorrere lo schermo in direzione verticale, lo dovrai portare a 24 righe);

2) porre i 3 bit meno significativi del registro interessato al valore massimo o minimo (dipende se lo scorrimento viene fatto dall' alto verso il basso o viceversa, da sinistra verso destra o da destra verso sinistra);

3) scrivere i dati nella parte nascosta dello schermo;

4) incrementare (o decrementare) il registro fino al valore massimo (o minimo);

5) riportare il registro al valore del passo due e far scorrere tutto lo schermo di un carattere intero nella direzione dello scorrimento; questa operazione deve essere fatta con una routine in linguaggio macchina, poichè in BASIC si vede nettamente che questa operazione equivale a fare 7 passi indietro più 8 in avanti,mentre deve sembrare sempre che sia un solo passo in avanti;

6) tornare al punto 3.

Vediamo un paio di esempi, che chiariscano meglio questi concetti, con i programmi GRAF23 e GRAF24.

```
1 REM GRAF23
10 POKE53280,0:POKE53281,0
15 PRINTCHR$(147)CHR$(154)
20 PRINT"CHE SCRITTA VUOI FAR SCORRERE ":PRINT
30 GETA$:IFA$=CHR$(13)ORLEN(B$)=253THEN50
40 B$=B$+A$:PRINTA$;:GOTO30
50 B$=B$+"  "
60 L=LEN(B$):PRINTCHR$(147)
65 POKE53270,PEEK(53270)AND247
70 FORI=1TOLEN(B$)
80 PRINTCHR$(19)" ";
100 PRINTCHR$(20);
103 POKE53270,(PEEK(53270)AND248)+7
```

```
105 PRINTCHR$(17)CHR$(157)MID$(B$,I,1)
110 FORJ=6TO0STEP-1
120 POKE53270,(PEEK(53270)AND248)+J
130 NEXT
140 NEXT
150 GOTO70
```

COMMENTO A GRAF23.
Linee 10-15: inizializza il video.
Linea 20: scrive la domanda.
Linee 30-40: ricevono una stringa che può essere lunga fino a 253 caratteri.
Linea 50: aggiunge 2 spazi alla stringa.
Linee 60-65: calcola la lunghezza della stringa, pulisce lo schermo e seleziona il modo a 38 colonne.
Linea 70: inizializza un ciclo.
Linea 80: posiziona il cursore sulla seconda colonna della prima linea.
Linea 100: cancella un carattere in modo da far scorrere tutta la prima linea a sinistra e riposizionare il cursore nell'angolo in alto a sinistra, pone 7 nei tre bit meno significativi del registro dello scorrimento fine orizzontale (7 = MAX a destra),sposta il cursore in basso e quindi a sinistra per posizionarlo sull'ultimo carattere della prima linea dove stampa l'i-esimo carattere della stringa.
Linee 110-130: ciclo che decrementa il contenuto dei 3 bit meno significativi del registro di scorrimento orizzontale fino a 0, spostando così le scritte gradualmente fino al massimo a sinistra.
Linea 140: chiude il ciclo iniziato nella linea 70.
Linea 150: salta nuovamente all'inizio del ciclo e ripete fino a che il programma non venga fermato con STOP o STOP/RESTORE.

Come vedi un programma completamente BASIC, non dà dei buoni risultati. La linea 100 andrà quindi sostituita con un'adeguata routine in linguaggio macchina, come la GRAF24, di cui riportiamo il listato commentato.

GRAF24

```
,C000 78        SEI
  ;disabilita l'interrupt
,C001 AD 12 D0 LDA $D012
,C004 C9 50     CMP #$50
,C006 D0 F9     BNE $C001
```

```
,C008 AD 11 D0 LDA $D011
,C00B 29 80    AND #$80
,C00D D0 F2    BNE $C001
  ;attende che il registro raster
  ;contenga $50
,C00F A9 14    LDA #$14
  ;carica nell'accumulatore $14 = 20
  ;ASCII di delete
,C011 20 CA F1 JSR $F1CA
  ;salta alla routine di output a video
,C014 AD 16 D0 LDA $D016
,C017 09 07    ORA #$07
,C019 8D 16 D0 STA $D016
  ;pone 7 nel registro dello smooth
  ;scrolling (7 = il massimo a destra)
,C01C 58       CLI
  ;riabilita l'interrupt
,C01D 60       RTS
  ;torna al basic
```

Il programma GRAF23, per lo scorrimento orizzontale, diventerà quindi il GRAF25.

```
1 REM GRAF25
5 FORI=49152TO49181:READA:POKEI,A:NEXT
10 POKE53280,0:POKE53281,0
15 PRINTCHR$(147)CHR$(154)
20 PRINT"CHE SCRITTA VUOI FAR SCORRERE ":PRINT
30 GETA$:IFA$=CHR$(13)ORLEN(B$)=253THEN50
40 B$=B$+A$:PRINTA$;:GOTO30
50 B$=B$+"  "
60 L=LEN(B$):PRINTCHR$(147)
65 POKE53270,PEEK(53270)AND247
70 FORI=1TOLEN(B$)
80 PRINTCHR$(19)" ";
100 SYS49152:PRINTCHR$(17)CHR$(157)MID$(B$,I,1)
110 FORJ=6TO0STEP-1
```

```
120 POKE53270,(PEEK(53270)AND248)+J
130 NEXT
140 NEXT
150 GOTO70
1000 DATA120,173,18,208,201,80,208,249
1010 DATA173,17,208,41,128,208,242,169
1020 DATA20,32,202,241,173,22,208,9
1030 DATA7,141,22,208,88,96
```

Le modifiche apportate a GRAF23, per ottenere GRAF25, sono indicate nel commento.

COMMENTO A GRAF25.
Linea 5: carica in memoria da 49152 (C000H) la routine assembler GRAF24.
Linea 100: sostituito ai comandi che cancellano e spostano le scritte a destra, il comando che salta alla routine in linguaggio macchina.
Linee 1000-1030: contengono i dati relativi alla routine GRAF24.

Per uno scorrimento fine verticale le cose si complicano ulteriormente: infatti, se chiediamo uno scorrimento fine mentre il registro di linea è in una posizione compresa tra l'inizio e la fine dello schermo, otteniamo uno spiacevole sfarfallare dello schermo e lo stesso vale, a maggior ragione, quando chiediamo un'operazione del tipo 7 passi indietro più 8 avanti. Per risolvere questi problemi bisogna quindi sincronizzare gli scorrimenti fini con il registro di linea e compiere lo scorrimento di 8 passi in maniera rapidissima. Purtroppo neppure una routine in linguaggio macchina riesce ad essere così veloce da far scorrere lo schermo mentre il registro di linea è fuori dallo schermo. Abbiamo quindi fatto una routine in linguaggio macchina, la GRAF26, che dispone di 2 pagine video: quando viene visualizzata la prima il BASIC provvede ad aggiungere una linea nuova nella parte nascosta della pagina video (usiamo 24 righe) e la routine in linguaggio macchina trasferisce nella seconda le scritte già posizionate una linea più in alto (questa operazione può essere fatta in maniera relativamente lenta poichè la seconda pagina non viene visualizzata). Al fatidico momento degli 8 passi avanti e dei 7 indietro non facciamo altro che cambiare pagina video e compiere solo 7 passi indietro. Essendo unica la mappa del colore non è stato possibile alternarne due e perciò tutte le scritte che si vedono scorrere saranno dello stesso colore.
Guardiamo prima la routine assembler GRAF26 e poi il programma GRAF27.

COMMENTO A GRAF26.
Dall'indirizzo 49152 (C000H) ha il compito di fare scorrimenti fini sincronizzati col registro di linea e, se è il caso (cioè ogni 8 passi), cambiare la mappa video.
Dall'indirizzo 49205 (C035H) inizia la routine che trasferisce una pagina nell'altra.

GRAF26

```
,C000 78        SEI
  ;maschera l'interrupt
,C001 AD 12 D0 LDA $D012
,C004 C9 FF     CMP #$FF
,C006 D0 F9     BNE $C001
,C008 AD 11 D0 LDA $D011
,C00B 29 80     AND #$80
,C00D D0 F2     BNE $C001
  ;attende che il VIC II stia disegnando
  ;la parte non visibile dello schermo
,C00F A5 02     LDA $02
,C011 D0 14     BNE $C027
  ;se non deve cambiare mappa video salta
,C013 AD 18 D0 LDA $D018
,C016 49 10     EOR #$10
,C018 8D 18 D0 STA $D018
  ;cambia mappa video nel VIC II
,C01B AD 88 02 LDA $0288
,C01E 49 04     EOR #$04
,C020 8D 88 02 STA $0288
  ;avvisa il sistema operativo
,C023 A9 08     LDA #$08
,C025 85 02     STA $02
  ;ricarica il contatore
,C027 C6 02     DEC $02
  ;decrementa il contatore (qui anche
  ;quando non cambia mappa video
,C029 AD 11 D0 LDA $D011
,C02C 29 F0     AND #$F0
,C02E 18        CLC
,C02F 65 02     ADC $02
,C031 8D 11 D0 STA $D011
```

```
  ;Pone nei 3 bit meno significativi del
  ;registro 17 del VIC II il contenuto del
  ;contatore Pone il video a 24 linee
,C034 60        RTS
  ;torna al basic
,C035 AD 88 02 LDA $0288
,C038 C9 3C     CMP #$3C
  ;guarda qual'e' la Pagina video corrente
,C03A D0 1D     BNE $C059
  ;se e' la Prima ($0380-03C0) salta
,C03C A9 3C     LDA #$3C
,C03E 85 FF     STA $FF
,C040 A9 28     LDA #$28
,C042 85 FE     STA $FE
,C044 A9 38     LDA #$38
,C046 85 FD     STA $FD
,C048 A9 00     LDA #$00
,C04A 85 FC     STA $FC
,C04C A9 3B     LDA #$3B
,C04E 8D 9E C0 STA $C09E
,C051 A9 C0     LDA #$C0
,C053 8D 9D C0 STA $C09D
  ;sistema i Puntatori Per trasferire
  ;la seconda Pagina
,C056 18        CLC
,C057 90 1A     BCC $C073
  ;salta sempre (abbiamo Preferito fare in
  ;questo modo Per rendere il Programma
  ;rilocabile con maggior facilita'
,C059 A9 38     LDA #$38
,C05B 85 FF     STA $FF
,C05D A9 28     LDA #$28
,C05F 85 FE     STA $FE
,C061 A9 3C     LDA #$3C
,C063 85 FD     STA $FD
,C065 A9 00     LDA #$00
,C067 85 FC     STA $FC
,C069 A9 3F     LDA #$3F
,C06B 8D 9E C0 STA $C09E
,C06E A9 C0     LDA #$C0
,C070 8D 9D C0 STA $C09D
```

```
  ;sistema i puntatori per trasferire
  ;la prima pagina
,C073 A0 00     LDY #$00
,C075 B1 FE     LDA ($FE),Y
,C077 91 FC     STA ($FC),Y
,C079 E6 FC     INC $FC
,C07B D0 02     BNE $C07F
,C07D E6 FD     INC $FD
,C07F E6 FE     INC $FE
,C081 D0 02     BNE $C085
,C083 E6 FF     INC $FF
,C085 A5 FD     LDA $FD
,C087 CD 9E C0 CMP $C09E
,C08A D0 E9     BNE $C075
,C08C A5 FC     LDA $FC
,C08E CD 9D C0 CMP $C09D
,C091 D0 E2     BNE $C075
  ;trasferisce la pagina video
,C093 A0 27     LDY #$27
  ;$27=40 numero di caratteri in una linea
,C095 A9 20     LDA #$20
  ;$20=32 codice ASCII dello spazio
,C097 91 FC     STA ($FC),Y
,C099 88        DEY
,C09A 10 FB     BPL $C097
  ;cancella l'ultima linea
,C09C 60        RTS
  ;torna al basic
```

```
1 REM GRAF27
10 POKE56,56:CLR
20 FORI=1TO40:SP$=SP$+CHR$(32):NEXT
30 G$=CHR$(19):FORI=1TO24:G$=G$+CHR$(17):NEXT
40 FORI=49152TO49308:READA:POKEI,A:NEXT
50 READN
60 DIMLN$(N-1)
70 FORI=0TON-1
80 READLN$(I)
90 IFLEN(LN$(I))>40THEN290
100 LN$(I)=LEFT$(SP$,(40-LEN(LN$(I)))/2)+LN$(I)
```

```
 110 NEXT
 120 POKE53281,14
 130 POKE648,56:PRINTCHR$(147)
 140 POKE648,60:PRINTCHR$(147)
 150 POKE53280,0:POKE53281,0
 160 POKE53272,(PEEK(53272)AND15)+240
 170 POKE2,7
 180 FORI=0TON-1
 190 PRINTG$;LN$(I);
 200 FORK=0TO20:NEXT
 210 SYS49205
 220 SYS49152
 230 FORJ=1TO7
 240 FORK=1TO60:NEXT
 250 SYS49152
 260 NEXTJ
 270 NEXTI
 280 GOTO180
 290 PRINT"LINEA "I+1" TROPPO LUNGA":END
 10000 DATA120,173,018,208,201,255,208,249
 10010 DATA173,017,208,041,128,208,242,165
 10020 DATA002,208,020,173,024,208,073,016
 10030 DATA141,024,208,173,136,002,073,004
 10040 DATA141,136,002,169,008,133,002,198
 10050 DATA002,173,017,208,041,240,024,101
 10060 DATA002,141,017,208,096,173,136,002
 10070 DATA201,060,208,029,169,060,133,255
 10080 DATA169,040,133,254,169,056,133,253
 10090 DATA169,000,133,252,169,059,141,158
 10100 DATA192,169,192,141,157,192,024,144
 10110 DATA026,169,056,133,255,169,040,133
 10120 DATA254,169,060,133,253,169,000,133
 10130 DATA252,169,063,141,158,192,169,192
 10140 DATA141,157,192,160,000,177,254,145
 10150 DATA252,230,252,208,002,230,253,230
 10160 DATA254,208,002,230,255,165,253,205
 10170 DATA158,192,208,233,165,252,205,157
 10180 DATA192,208,226,160,039,169,032,145
 10190 DATA252,136,016,251,096
 20000 DATA10
```

```
20010 DATA"********************"
20020 DATA"*                  *"
20030 DATA"*   COMMODORE 64   *"
20040 DATA"*                  *"
20050 DATA"* SMOOTH SCROLLING *"
20060 DATA"*                  *"
20070 DATA"********************"
20080 DATA"","","""
```

COMMMENTO A GRAF27.
Linea 10: pone la fine della memoria a 14336 (3800H) dove inizierà la prima pagina video, la seconda inizierà in 15360 (3C00H).
Linea 20: riempie la variabile SP$ con 40 spazi.
Linea 30: la variabile contiene HOME e 24 CURSORE GIU'; quando verrà stampata il cursore si posizionerà all'inizio dell' ultima linea.
Linea 40: carica in memoria la routine in linguaggio macchina.
Linea 50: legge il numero di stringhe che dovrà visualizzare.
Linea 60: dimensiona un vettore di stringhe che conterrà le N stringhe richieste.
Linee 70-110: riempono la matrice controllando che le stringhe non siano più lunghe di 40 caratteri e centrandole.
Linea 120: schermo blù chiaro.
Linea 130: avvisa il sistema operativo che la pagina video è ora in 14336 (3800H) (vedi Paragrafo 1.3.2) e pulisce lo schermo. Pulendo lo schermo mentre questo è blu chiaro il COMMODORE 64 riempie automaticamente la mappa del colore di blù chiaro in modo che, portando lo schermo a nero, basterà mettere il codice del carattere nella mappa video e questo comparirà di colore blu chiaro.
Linea 140: avvisa il sistema operativo che la pagina video è ora in 15360 (3C00H) e pulisce lo schermo.
Linea 150: schermo e bordo neri.
Linea 160: sposta, nel VIC II, la pagina video a 15360 (3C00H).
Linea 170: inizializza il contatore della routine assembler.
Linea 180: inizializza un ciclo.
Linea 190: scrive sull'ultima linea la i-esima stringa del vettore.
Linea 200: attende perchè lo scorrimento non sia troppo veloce.
Linea 210: salta alla routine che trasferisce la pagina video.
Linea 220: salta alla routine che fa scorrere lo schermo.
Linee 230-260: per 7 volte fa uno scorrimento e attende (il ritardo qui è maggiore per compensare il tempo impiegato dalla routine di trasferimento della mappa video).

Linea 270: chiude il ciclo iniziato nella linea 180.
Linea 280: ricomincia fino a che il programma non viene fermato.
Linee 10000-10190: contengono i dati relativi alla routine in linguaggio macchina GRAF26.
Linea 20000: numero delle stringhe che si vogliono visualizzare.
Linee 20010-20080: stringhe che si vogliono visualizzare.

CAPITOLO 5

UN PICCOLO SISTEMA GRAFICO

Come abbiamo già visto nei Capitoli 3 e 4, è necessario lavorare un po' con il linguaggio macchina per poter sfruttare al meglio le capacità grafiche del COMMODORE 64. Nel Capitolo 3 ti abbiamo mostrato due routine per poter cancellare la pagina grafica e tracciarvi dei punti.
Ci sono però un paio di problemi che forse avrai avuto modo di constatare: lo spazio che rimane per il programma è piuttosto poco e, se si verificano errori mentre il video è in modo grafico, non si riescono a leggere, a meno di non effettuare alla cieca le POKE corrette.
Entrambi i problemi sono dovuti al fatto che, per mantenere il massimo di semplicità nelle spiegazioni, abbiamo voluto lavorare nel banco 0, in modo da non avere problemi di passaggio da un banco all'altro, e non abbiamo voluto inoltrarci in argomenti che non hanno niente a che vedere con la grafica e avrebbero solo confuso le idee.
A questo punto del volume, però, tu hai ormai preso confidenza con tutti gli argomenti trattati e qualche complicazione in più non ti spaventerà.
Quello che ti proponiamo ora è un piccolo sistema grafico in alta risoluzione, appena più complesso di quello del capitolo 3, che ruba al BASIC solo gli 8K di RAM da 32768 a 40959, lasciandoti quindi 30170 byte liberi per i programmi.

5.1 LA PAGINA GRAFICA

Il primo problema da affrontare è stato quello di dove prendere gli ottomila byte necessari per lavorare in grafica, senza rubare troppa memoria al BASIC, e lasciando spazio sufficiente anche per le nuove routine.
La scelta, quasi obbligata, è stata di mettere la pagina grafica SOTTO una delle due ROM, quella del BASIC.
Abbiamo preferito quest'area per due motivi: intanto, premendo RUN/STOP + RESTORE la pagina grafica da 57344 ($E000) viene sporcata, inoltre se vorrai aggiungere le routine di lettura e salvataggio della pagina grafica, potrai farlo molto più comodamente se dovrai disabilitare la RAM del BASIC piuttosto che quella del KERNAL (che contiene appunto le routine di Input/Output!).

Il banco di memoria interessato è il numero 2 (32768-49151) e qui sorge un altro problema: il VIC II "vede" agli indirizzi da 36846 a 40959 la ROM generatrice dei caratteri. Non è quindi possibile posizionarvi la mappa video che fa da memoria COLORE per la pagina grafica.
Soluzione: utilizzare gli indirizzi da 32768 a 33791 per la pagina video (e i puntatori agli sprite) e il resto della memoria, compresa quella che il VIC II considera ROM ma la CPU del COMMODORE 64 usa come RAM, per memorizzare le nostre routine.
La mappa di memoria che ne risulta è schematizzata di seguito:

esadecim.	decimale	uso
0000-7FFF	(-32767)	RAM BASIC
8000-83FF	(32768-33791)	MAPPA VIDEO/COLORE
8400-9FFF	(33792-40959)	routine GRAFICHE
A000-BF3F	(40960-48959)	BIT-MAP GRAFICA

Le nostre nuove routine si preoccuperanno di modificare il banco in uso ed anche di fare entrare ed uscire il VIC II dal modo grafico, diminuendo il numero di istruzioni POKE da utilizzare nei programmi.

5.2 L'AREA DEI PARAMETRI

Per il passaggio dei parametri alle routine abbiamo pensato di utilizzare alcune locazioni nelle prime pagine della memoria, sfruttate solo dalle varie espansioni del BASIC in circolazione. Quello che segue è l'elenco degli indirizzi utilizzati dalle routine. Rimane comunque spazio per gli ampliamenti che tu riterrai opportuni.

Esadec.	Dec.	Significato
02A8	(680)	- carattere di riempimento per la routine di pulizia dello schermo
02A9	(681)	- byte con cui riempire la mappa colore
02AA	(682)	- byte di salvataggio per il banco in uso al momento del passaggio in grafica
02AB	(683)	- byte di salvataggio per gli indirizzi della mappa video e dei caratteri al momento del passaggio in grafica
02AC	(684)	- MODO per la routine di PLOT: 0 : cancella 1 : traccia 2..255 : inverte
02AD	(685)	- byte per memorizzare la maschera utilizzata da PLOT per disegnare/cancellare un punto
02AE	(686)	- X1 (byte meno significativo)
02AF	(687)	- X1 (byte più significativo)
02B0	(688)	- Y1
02B1	(689)	- X2 (byte meno significativo)
02B2	(690)	- X2 (byte più significativo)
02B3	(691)	- Y2

5.3 INGRESSO NEL MODO GRAFICO AD ALTA RISOLUZIONE

Non ripetiamo qui le spiegazioni del Capitolo 3, riguardale se necessario, ma ti proponiamo direttamente il primo blocco di routine con una breve spiegazione. Quelle che seguono sono le routine (scritte sulla falsariga di GRAF12) che puliscono la pagina grafica (o meglio, la riempiono con il byte che tu avrai posto all'indirizzo 680=$02A8), riempiono la mappa colore con il valore nel byte 681 (=$02A9), entrano ed escono dal modo grafico ed effettuano una piccola variazione all'interprete BASIC così che, in caso di errore, si torni automaticamente al video in modo TESTO.
La variazione effettuata consiste nella modifica del vettore in 770-771 ($0302-$0303), quello del WARM-START del BASIC (Per intenderci, è il puntatore alla routine che dopo la stampa della scritta READY attende un comando). Esso viene fatto puntare ad una routine che, se necessario, esce dal modo grafico ripristinando i valori corretti di banco e video e quindi salta alla normale routine di WARM-START.

```
;
;  routine di riempimento mappa alta-risoluzione
;  $8400=33792
;
,8400   a9  a0      lda  #$a0     ; il puntatore in pagina 0 $FB-$FC è inizia-
,8402   85  fc      sta  $fc      ; lizzato a $A000 (40960) (inizio pagina gra-
,8404   a9  00      lda  #$00     ; fica)
,8406   85  fb      sta  $fb      ; (inizio pagina grafica)
,8408   a2  20      ldx  #$20     ; deve riempire 32 pagine da 256 byte
,840a   ad  a8  02  lda  $02a8    ; con il valore in 680
,840d   a0  00      ldy  #$00     ; riempie una pagina
,840f   91  fb      sta  ($fb),y  ;
,8411   c8          iny           ;
,8412   d0  fb      bne $840f     ;
,8414   e6  fc      inc  $fc      ; punta alla successiva
,8416   ca          dex           ; se non era l'ultima prosegue
,8417   d0  f4      bne $840d     ;
,8419   60          rts
;
;  routine riempimento mappa colore
;  $841A=33818
;
,841a   ad  a9  02  lda  $02a9    ; legge il parametro in 681
,841d   a0  00      ldy  #$00     ; e riempie la mappa colore
,841f   99  00  80  sta  $8000,y  ; (4 pagine da 256 byte)
,8422   99  00  81  sta  $8100,y  ;
,8425   99  00  82  sta  $8200,y  ;
,8428   99  00  83  sta  $8300,y  ;
,842b   c8          iny
,842c   d0  f1      bne $841f
,842e   60          rts
;
;  entra in modo grafico
;  $842F=33839
;
,842f   20  00  84  jsr  $8400    ; riempie schermo bit-map
;  $8432=33842
,8432   20  1a  84  jsr  $841a    ; e riempie la mappa colore (4 pagine da 256
;                                   byte)
```

```
;  8435=33845          punto di ingresso per non variare
;                      i contenuti dello schermo
;
,8435   ad  00  dd  lda  $dd00    ; salva il valore del banco in uso
,8438   8d  aa  02  sta  $02aa    ;
,843b   29  fc      and  #$fc     ; e seleziona il banco #
,843d   09  01      ora  #$01     ;
,843f   8d  00  dd  sta  $dd00    ;
,8442   ad  18  d0  lda  $d018    ; salva il puntatore alle mappe video e carat-
,8445   8d  ab  02  sta  $02ab    ; teri
,8448   a9  08      lda  #$08     ; e pone video in 32768
,8450   09  20      ora  #$20     ; e bitmap in 40960
,844d   ad  11  d0  lda  $d011    ; attiva il modo grafico
,844a   8d  18  d0  sta  $d018    ;
,8452   8d  11  d0  sta  $d011    ;
,8455   60          rts
;  esce dal modo grafico
;  8456=33878
;
,8456   ad  aa  02  lda  $02aa    ; ripristina il banco
,8459   8d  00  dd  sta  $dd00    ;
,845c   ad  ab  02  lda  $02ab    ; le mappe video e caratteri
,845f   8d  18  d0  sta  $d018    ;
,8462   ad  11  d0  lda  $d011    ; e disattiva il modo grafico
,8465   29  df      and  #$df     ;
,8467   8d  11  d0  sta  $d011    ;
,846a   60          rts
;
;  nuova routine di
;  WARM-START
;
,846b   a9  20      lda  #$20     ; se il video è in modo grafico
,846d   2d  11  d0  and  $d011    ;
,8470   f0  03      beq  $8475
,8472   20  56  84  jsr  $8456    ; torna in modo testo
,8475   4c  83  a4  jmp  $a483    ; poi salta alla normale routine di WARM-
                                  ; START
;
;  installa la nuova
;  routine di WARM-START
;  $8478=33912
;
```

```
,8478   78          sei          ; disabilita interruzioni
,8479   a2  6b      ldx #$6b     ; modifica il vettore di WARM-START
,847b   a0  84      ldy #$84     ;
,847d   8e  02  03  stx $0302    ;
,8480   8c  03  03  sty $0303    ;
,8483   58          cli          ; riabilita interruzioni
,8484   60          rts          ; ed esce
```

Ovviamente, prima di iniziare a lavorare con le routine grafiche occorre aver riservato loro la memoria, caricato le routine e installato la nuova routine di WARM-START. È quello che fa il caricatore BASIC (GRAF29) riportato di seguito.

GRAF29

```
10      REM GRAF29
20      REM CARICA E INSTALLA
30      REM  LA PRIMA  PARTE
40      REM   DELLE ROUTINE
50      REM
10000 POKE 56,128:POKE644,128:CLR
10010 I=33792
10020 READ A$: IF A$="*" THEN SYS 33912: NEW
10030 POKE I,VAL(A$):I=I+1:GOTO10020
10040 DATA169,160,133,252,169,0,133,251
10050 DATA162,32,173,168,2,160,0,145
10060 DATA251,200,208,251,230,252,202,208
10070 DATA244,96,173,169,2,160,0,153
10080 DATA0,128,153,0,129,153,0,130
10090 DATA153,0,131,200,208,241,96,32
10100 DATA0,132,32,26,132,173,0,221
10110 DATA141,170,2,41,252,9,1,141
10120 DATA0,221,173,24,208,141,171,2
10130 DATA169,8,141,24,208,173,17,208
10140 DATA9,32,141,17,208,96,173,170
10150 DATA2,141,0,221,173,171,2,141
10160 DATA24,208,173,17,208,41,223,141
10170 DATA17,208,96,169,32,45,17,208
10180 DATA240,3,32,86,132,76,131,164
10190 DATA120,162,107,160,132,142,2,3
10200 DATA140,3,3,89,96,*
```

Una volta letto ed eseguito GRAF 29 puoi provare i due esempi che seguono. Il primo ti mostra cosa succede entrando in grafica e richiamando solo la routine di riempimento della bit-map ogni volta con un carattere diverso.

GRAF30

```
5  REM GRAF30
10 SYS 33912
20 POKE 681,22:POKE680,0: SYS 33839
30 FOR K=0 TO 255
40 POKE 680,K: SYS 33792
50 NEXT
```

In questo secondo esempio puoi vedere l'effetto che si ottiene cambiando invece solo i colori.

GRAF31

```
5  REM GRAF31
10 SYS 33912
20 POKE 681,0:POKE680,3: SYS 33839
30 FOR K=0 TO 255
40 POKE 681,K: SYS 33818
50 GETA$:IFA$=""THEN50
60 NEXT
```

5.4 PLOT/UNPLOT/INVERT

Ora che possiamo inizializzare i contenuti del nostro schermo grafico ci serve una nuova routine per disegnarvi dei punti.
Questa nuova routine si differenzia da quella del capitolo 3 per i seguenti motivi:
— ovviamente, disegna nella pagina grafica in $A000;
— per semplificare il calcolo dell'indirizzo del byte da modificare e del byte da usare come maschera per disegnare il punto, questa routine si serve di tabelle in memoria. Come puoi notare dal listato ciò riduce anche l'occupazione di memoria della routine;
— a seconda del contenuto del byte di indirizzo 684 (=$02AC) la routine disegna o cancella un punto oppure inverte quello già presente nello schermo.

```
:8486   80  40  20  10  08  04  02  01   ; dati per determinare il byte da
                                         ; usare come maschera
                                         ;
:848e   00  a0  40  a1  80  a2  c0  a3   ; indirizzo del primo byte di ogni
                                         ; gruppo di 8 linee
:8496   00  a5  40  a6  80  a7  c0  a8   ; Questa tabella evita la moltiplica-
:849e   00  aa  40  ab  80  ac  c0  ad   ; zione per 320
:84a6   00  af  40  b0  80  b1  c0  b2   ;
:84ae   00  b4  40  b5  80  b6  c0  b7   ;
:84b6   00  b9  40  ba  80  bb  c0  bc   ;
:84be   00  be                           ;
;
;  determina il byte maschera per l'operazione di PLOT
;  $84C0=33984
,84c0   ad  ae  02  lda $02ae            ; il valore dei 3 bit meno significativi di
,84c3   29  07      and #$07             ; X1 è usato come indice nella tabella
,84c5   aa          tax                  ; che inizia in $8486
,84c6   bd  86  84  lda $8486,x          ;
                                         ;
,84c9   8d  ad  02  sta $02ad            ; salva il valore letto
,84cc   90  01      bcc $84cf            ; se il CARRY è a 1 termina
,84ce   60          rts                  ;
;
;  calcola l'indirizzo del byte che contiene
;  la descrizione del punto di coordinate X1 e Y1
;  in $FB-$FC
;  $84CF=33999
,84cf   ad  af  02  lda $02af            ; in $FB-$FC va X1 con i suoi 3 bit
,84d2   85  fc      sta $fc              ; meno significativi posti a 0
,84d4   ad  ae  02  lda $02ae            ;
,84d7   29  f8      and #$f8             ;
,84d9   85  fb      sta $fb              ;
  ;  INT(X1/8)*8
,84db   a9  c7      lda #$c7             ; corregge Y1 in 199-Y1
,84dd   38          sec                  ;
,84de   ed  b0  02  sbc $02b0            ;
,84e1   aa          tax                  ; e lo salva nel reg. X
,84e2   4a          lsr                  ; calcola l'indice per la tabella in $848e
,84e3   4a          lsr                  ;
```

```
,84e4   4a          lsr              ;
,84e5   0a          asl              ;
,84e6   a8          tay
,84e7   b9  8e  84  lda  $848e,y     ; legge l'indirizzo dalla tabella e lo som-
,84ea   18          clc              ; ma a quello attuale
,84eb   65  fb      adc  $fb         ;
,84ed   85  fb      sta  $fb         ;
,84ef   b9  8f  84  lda  $848f,y
,84f2   65  fc      adc  $fc
,84f4   85  fc      sta  $fc
  ;  INT(X1/8)*8+320*INT(Y1/8)+40960
,84f6   8a          txa              ; recupera 199-Y1
,84f7   18          clc              ; somma i tre bit meno significativi di
,84f8   29  07      and  #$07        ; Y1 (corretto)
,84fa   65  gb      adc  $fb         ;
,84fc   85  fb      sta  $fb         ;
,84fe   90  02      bcc  $8502       ;
,8500   e6  fc      inc  $fc         ;
,8502   60          rts              ; ed ha il risultato voluto
  ;  INT(X1/8)*8+320*INT(Y1/8)+40960+Y1—(INT(Y1/8)*8)
;
;  routine di PLOT
;  $8503=34051
,8503   18          clc              ; determina maschera ed indirizzo da
,8504   20  c0  84  jsr  $84C0       ; modificare
,8507   a0  00      ldy  #$00        ; (reg. Y come indice)
,8509   78          sei              ; disabilita interruzioni
,850a   a9  fe      lda  #$fe        ; e la ROM del BASIC
,850c   25  01      and  $01         ;
,850e   85  01      sta  $01         ;
,8510   aa          tax              ; legge byte da alterare e lo salva in $02
,8511   b1  fb      lda  ($fb),y     ;
,8513   85  02      sta  $02         ;
,8515   8a          txa              ; riabilita BASIC
,8516   09  01      ora  #$01        ;
,8518   85  01      sta  $01         ;
,851a   58          cli              ; ed interruzioni
,851b   ad  ad  02  lda  $02ad       ; in ACC il byte maschera
,851e   ae  ac  02  ldx  $02ac       ; in reg. X il modo
,8521   d0  07      bne  $852a       ; salta se reg. X <> 0
;
```

```
;  cancella un punto
;
,8523   49  ff      eor #$ff        ; mette a 0 il bit del punto scelto
,8525   25  02      and $02         ;
,8527   91  fb      sta ($fb),y     ; e modifica la memoria
,8529   60          rts
,852a   e0  01      cpx #$01        ; se il modo è <> 1 salta
,852c   d0  05      bne $8533       ;
;
;  disegna un punto
;
,852e   05  02      ora $02         ; mette a 1 il bit del punto
,8530   91  fb      sta ($fb),y     ; e modifica la memoria
,8532   60          rts             ;
;
;  inverte un punto
;
,8533   45  02      eor $02         ; inverte il bit del punto
,8535   91  fb      sta ($fb),y     ; e modifica la memoria
,8537   60          rts             ;
```

Il caricatore che segue va utilizzato dopo quello delle routine di pulizia dello schermo.

GRAF33

```
10      REM GRAF33
20      REM CARICA LA ROUTINE
30      REM PLOT/UNPLOT/INVERT
40      REM
10000 REM
10010 I=33926
10020 READ A$: IF A$="*" THEN NEW
10030 POKE I,VAL(A$):I=I+1:GOTO10020
10040 DATA128,64,32,16,8,4,2,1
10050 DATA0,160,64,161,128,162,192,163
10060 DATA0,165,64,166,128,167,192,168
10070 DATA0,170,64,171,128,172,192,173
10080 DATA0,175,64,176,128,177,192,178
10090 DATA0,180,64,181,128,182,192,183
10100 DATA0,185,64,186,128,187,192,188
10110 DATA0,190,173,174,2,41,7,170
10120 DATA189,134,132,141,173,2,144,1
10130 DATA96,173,175,2,133,252,173,174
10140 DATA2,41,248,133,251,169,199,56
10150 DATA237,176,2,170,74,74,74,10
10160 DATA168,185,142,132,24,101,251,133
10170 DATA251,185,143,132,101,252,133,252
10180 DATA138,24,41,7,101,251,133,251
10190 DATA144,2,230,252,96,24,32,192
10200 DATA132,160,0,120,169,254,37,1
10210 DATA133,1,170,177,251,133,2,138
10220 DATA9,1,133,1,88,173,173,2
10230 DATA174,172,2,208,7,73,255,37
10240 DATA2,145,251,96,224,1,208,5
10250 DATA5,2,145,251,96,69,2,145
10260 DATA251,96,*
```

Dopo aver caricato ed eseguito GRAF28 e GRAF33 puoi provare a modificare i programmi che disegnano le funzioni visti nel capitolo 3, non avrai molti problemi.

Come brevissimo esempio, per controllare che tutto funzioni, puoi provare il programma che segue che traccia le due diagonali dello schermo.

GRAF34

```
5  REM GRAF34
8  REM TRACCIA LE DIAGONALI DELLO SCHERMO
10 SYS 33912
20 POKE680,0:POKE681,16:SYS 33839
30 POKE 684,1
40 FOR X= 0 TO 319
50 POKE 686,XAND255:POKE687,X/256:POKE688,X*.625
60 SYS 34051
70 POKE 686,XAND255:POKE687,X/256:POKE688,200-X*.625
80 SYS 34051
90 NEXT
95 GETA$: IF A$="" THEN 95
```

5.5 LINEE ORIZZONTALI E VERTICALI

Fino a qui non è che abbiamo aggiunto molto alle vecchie routine, a parte la possibilità di cancellare o invertire punti.
Con poca fatica, però, possiamo ora crearci un paio di routine che traccino linee orizzontali e verticali.
Entrano in gioco adesso le coordinate X2 ed Y2 agl'indirizzi di memoria 689-691 ($02B1-$02B3).
Basta semplicemente creare un ciclo che mantenga fissa una coordinata ed incrementi l'altra finché non si raggiunge il limite prefissato (X2 o Y2); prova prima a farlo in BASIC, quindi esamina e prova le due routine che seguono:

GRAF35

```
;
;  VLINE traccia linee verticali
;  da X1, Y1 ad X1, Y2
;  $8540=34112
,8540   ad  b0  02  lda  $02b0      ; salva Y1 in $FD
,8543   85  fd      sta  $fd        ;
,8545   20  03  85  jsr  $8503      ; PLOT (X1, Y1)
,8548   ad  b0  02  lda  $02b0      ; se Y1 è uguale oppure maggiore di Y2,
,854b   cd  b3  02  cmp$02b3        ; fine
,854e   f0  08      beq $8558       ;
,8550   b0  06      bcs $8558       ;
,8552   ee  b0  02  inc  $02b0      ; altrimenti incrementa Y1
,8555   18          clc             ; e traccia il punto seguente
,8556   90  ed      bcc $8545       ;
;
,8558   a5  fd      lda  $fd        ; ripristina Y1
,855a   8d  b0  02  sta  $02b0      ;
,855d   60          rts             ; e termina
;
;  HLINE traccia linee orizzontali
;  da X1, Y1 ad X2, Y1
;  $855E=34142
,855e   ad  ae  02  lda  $02ae      ; salva X1 in $FD-$FE
,8561   85  fd      sta  $fd        ;
,8563   ad  af  02  lda  $02af      ;
,8566   85  fe      sta  $fe        ;
,8568   20  03  85  jsr  $8503      ; PLOT(X1, Y1)
,856b   ad  af  02  lda  $02af      ; confronta i byte alti di X1 ed X2
,856e   cd  b2  02  cmp$02b2        ;
,8571   f0  04      beq $8577       ; se sono uguali prosegue il confronto
                                    ;
,8573   b0  17      bcs $858c       ; se X1>X2 allora termina
,8575   90  0a      bcc $8581       ; se X1<X2 salta ad incrementare X1
                                    ;
,8577   ad  ae  02  lda  $02ae      ; confronta i byte bassi di  X1 ed X2
,857a   cd  b1  02  cmp$02b1        ;
,857d   f0  0d      beq $858c       ; se sono uguali o X1>X2 allora termi-
,857f   b0  0b      bcs $858c       ; na
,8581   ee  ae  02  inc  $02ae      ; incrementa X1
,8584   d0  e2      bne $8568       ;
```

```
,8586   ee  af  02  inc  $02af      ;
,8589   18          clc            ; e traccia il punto seguente
,858a   90  dc      bcc  $8568     ;
;
,858c   a5  fd      lda  $fd       ; ripristina X1
,858e   8d  ae  02  sta  $02ae     ;
,8591   a5  fe      lda  $fe       ;
,8593   8d  af  02  sta  $02af     ;
,8596   60          rts  $84C0     ; e termina
```

Ecco il caricatore BASIC per le due routine:

GRAF36

```
10      REM GRAF36
20      REM CARICA ROUTINE
30      REM HLINE &  VLINE
40      REM
10000   REM
10010   I=34112
10020   READ A$: IF A$="*" THEN NEW
10030   POKE I,VAL(A$):I=I+1:GOTO10020
10040   DATA173,176,2,133,253,32,3,133
10050   DATA173,176,2,205,179,2,240,8
10060   DATA176,6,238,176,2,24,144,237
10070   DATA165,253,141,176,2,96,173,174
10080   DATA2,133,253,173,175,2,133,254
10090   DATA32,3,133,173,175,2,205,178
10100   DATA2,240,4,176,23,144,10,173
10110   DATA174,2,205,177,2,240,13,176
10120   DATA11,238,174,2,208,226,238,175
10130   DATA2,24,144,220,165,253,141,174
10140   DATA2,165,254,141,175,2,96,*
```

Poter tracciare solo linee non oblique serve a poco, ma si possono ottenere comunque effetti piacevoli.
Carica ed esegui GRAF28, GRAF 33 e GRAF35, poi prova il programma che segue il quale traccia una figura basata su due sinusoidi e ne evidenzia metà, invertendone il disegno usando la routine di HLINE con MODO=2:

GRAF37

```
5  REM GRAF37
10 SYS 33912
20 POKE680,0:POKE681,22:SYS 33839
30 POKE 684,1
40 FOR X=0 TO 319 STEP 4
50 POKE 686,X AND255:POKE687,X/256
55 POKE 688,50+30*SIN(X/17)
56 POKE 691,140+50*SIN(X/9)
60 SYS 34112
70 NEXT
100 POKE 684,2
110 FOR Y=199 TO 85 STEP -1
120 POKE 686,0:POKE687,0:POKE688,Y
130 POKE 689,63:POKE690,1
140 SYS 34142
150 NEXT
160 GETA$:IFA$="" THEN 160
```

5.6 IDEE

Con poche linee di codice puoi ottenere, a partire da HLINE e VLINE, almeno due routine: tracciamento di rettangoli pieni e di rettangoli vuoti.
Leggendo il volume 4 puoi imparare come incorporare i parametri per una routine in linguaggio macchina. nella SYS che usi per richiamarla.
Così come puoi trovare le informazioni necessarie per creare due routine che ti permettano di leggere e scrivere su disco o cassetta il contenuto della pagina grafica.
Per quanto riguarda il tracciamento di linee e curve, ti consigliamo di leggere qualche testo di grafica edito dalla JACKSON. Le routine più importanti, quelle che variano da computer a computer, per intenderci, sono già a tua disposizione.
Lo spazio che è rimasto (quasi 6K) è in grado di contenere le routine suggerite in questo paragrafo. E anche di più, buon lavoro.

INDICE

Capitolo 1 - ADATTATORE TELEMATICO 1

Capitolo 2 - GLI SPRITE
2.1 Cosa è uno sprite 7
2.2 Movimento di uno sprite 12
2.3 Altri esempi di animazione 23
2.4 Sprite multicolore 36
2.5 Riassunto delle caratteristiche degli sprite 44

Capitolo 3 - IL SUONO
3.1 Perché il suono 49
3.1.1 Come si genera un suono 49
3.1.2 Onde sonore, frequenza, volume e timbro 51
3.1.3 Suoni con il Commodore 64 52
3.1.4 Forme d'onda e ADSR, il timbro 54
3.2 Programmi sonori 62
3.3 L'uso dei filtri 68
3.3.1 La risonanza 73
3.4 Ancora di più 75
3.4.1 Modulazione ad anello e sincronizzazione 77
3.4.2 Cambiamenti dinamici del suono 78
3.5 Per concludere 79
3.6 Valori delle note 87

Capitolo 4 - I REGISTRI DEL VIC II
E I REGISTRI DEL SID
4.1 I registri del VIC II 91
4.2 I registri del SID 95

CAPITOLO 1

ADATTATORE TELEMATICO

Inserendo nella *porta di espansione* (posta sul retro del COMMODORE 64 a sinistra) l'ADATTATORE TELEMATICO 6499 e l'apposita spina tripolare passante tra la spina dell'apparecchio telefonico e la relativa presa SIP il tuo calcolatore diventa una *stazione telematica.*

Tramite l'adattatore telematico puoi collegarti, utilizzando la tua linea telefonica, con VIDEOTEL, il Servizio Telematico Pubblico, che ti permette oggi di accedere a 250 Banche Dati per ottenere informazioni in tempo reale inerenti a economia, finanza, istituti di credito, agricoltura, scuole, trasporti, turismo, tempo libero, situazione metereologica, e altro. Inoltre tutti gli utenti VIDEOTEL possono utilizzare il servizio di CASELLA POSTALE per lo scambio di messaggi.
Tramite l'adattatore telematico ti è possibile anche consultare le PAGINE GIALLE ELETTRONICHE, che forniscono utili informazioni commerciali.

La normale rete telefonica commutata funziona per questa applicazione con i due standard previsti V21 e V23. V21 funziona a 300 baud sia in ricezione che in trasmissione. V23 funziona a 1200 baud in ricezione e 75 in trasmissione.

```
1 REM PER MEMORIZZARE
2 REM SAVE"O:CALL EDIT",8
3 CD$="CELINTRUBRICAMK"
4 SP$="                    ":REM 16 SPAZI
5 CS$=CHR$(147):REM PULIZIA SCHERMO
6 D$=CHR$(17):REM CURSORE GIU'
7 PRINT CHR$(14);CHR$(8);
8 REM CHIAMATA DIRECTORY PER EDIT
9 OPEN 15,8,15,"IO"
10 REM ESISTE IL FILE?
11 OPEN 8,8,8,"O:"+CD$+",S,R"
12 INPUT#15,E,E$,T,S
13 CLOSE 8
14 IF E=0 THEN 17
15 OPEN 8,8,8,"O:"+CD$+",S,W"
16 GOTO 18
17 OPEN 8,8,8,"O:"+CD$+",S,A"
18 REM AGGIUNTA ENTRATE
19 PRINT CS$;D$;D$;"VIEWDATA/TERMINAL/EXIT (V,T,X) ?";
20 GET Q$:IF Q$="" THEN 20
21 IF Q$="X" THEN 49
22 IF Q$="V" THEN PRINT Q$:TS=32:GOTO 25
23 IF Q$="T" THEN PRINT Q$:TS=64:GOTO 25
24 GOTO 20
25 PRINT D$;D$;"NOME SERVIZIO (12 CARATTERI) ";
26 INPUT SN$
27 PRINT D$;D$;"NUMERO TELEFONO ";
28 INPUT NO$
29 IF TS=32 THEN P1=0:P2=0:P3=0:GOTO 43
30 PRINT D$;D$;"(A) 1200/75 (B) 300/300 ? ";
31 GET Q$:IF (Q$<>"A") AND (Q$<>"B") THEN 31
32 PRINT Q$:P1=1:IF Q$="B" THEN P1=3
33 PRINT D$;D$;"PARITA' (1-5) ? ";
34 GET Q$:IF (Q$<"1") OR (Q$>"5") THEN 34
35 P2=VAL(Q$)
36 PRINT Q$:PRINT D$;D$;"ECO LOCALE (Y/N) ?";
37 GOSUB 54
38 P3=0:IF Y$="Y" THEN P3=128
39 PRINT Y$:PRINT D$;D$;"AUTO LINE FEED (Y/N) ?";
40 GOSUB 54
41 IF Y$="Y" THEN P3=P3+64
42 PRINT Y$
43 PRINT#8,CHR$(TS);
44 PRINT#8,LEFT$(SN$+SP$,12);
45 PRINT#8,LEFT$(NO$+CHR$(0)+SP$,16);
46 PRINT#8,CHR$(P1);CHR$(P2);CHR$(P3);
47 INPUT#15,E,E$,T,S
48 IF E=0 THEN 19
49 CLOSE 8
50 PRINT CS$;"FINISHED":PRINT:PRINT
```

```
51 FOR L=1 TO 8:READ X:POKE 49151+L,X:NEXT
52 SYS 49152
53 DATA 169,62,141,3,222,108,252,255
54 GET Y$:IF (Y$<>"Y") AND (Y$<>"N") THEN 54
55 RETURN
```

L'adattatore deve essere inserito nella porta di espansione a calcolatore spento; dopo l'accensione del COMMODORE 64 compare sul video il menu principale:

MENU PRINCIPALE

F1: Agenda su disco
F2: Carica opzioni
F3: Esegue opzioni
F4: Modo disco
F5: Videotel
F6: Selezione periferiche
F7: Terminale scrolling
F8: Ritorno al Basic

e puoi scegliere la funzione che desideri.

FUNZIONE F1: Agenda su disco

Puoi creare su disco un'*agenda elettronica* per memorizzare i numeri di telefono delle Banche Dati. Per ogni Banca Dati puoi memorizzare, oltre al numero di telefono, anche il nome del servizio fornito, i parametri di connessione e un commento.
Per poter creare questa agenda puoi utilizzare il programma CALL EDIT che presentiamo; esso serve per creare ed aggiornare il file sequenziale di nome CELINTRUBRICAMK, che viene utilizzato come agenda. Per utilizzare il programma devi passare al BASIC con la funzione F8. Il programma CALL EDIT termina caricando ed eseguendo una breve routine in linguaggio macchina che restituisce il controllo all'adattatore telemmatico.

Programma D.1 CALL EDIT

Il programma in linguaggio assembler che abbiamo utilizzato, caricandoi valori decimali corrispondenti nei byte da 49152 a 42159, è il seguente:

Assembler	Esadecimale			Decimale		
LDA #$3E	A9	3E		169	62	
STA $DE0E	8D	03	DE	141	3	222
JMP $FFFC	6C	FC	FF	108	252	255

FUNZIONE F2: Carica opzioni

Con questa funzione puoi caricare dalla memoria di massa software per l'aggiornamento dei programmi residenti in ROM, senza dover intervenire sull'adattatore.

FUNZIONE F3: Esegue opzioni

È stata prevista questa funzione per potersi servire delle nuove opzioni aggiuntive caricate dalla memoria di massa con la funzione F2.

FUNZIONE F4: Modo disco

Compare nella prima linea del video il messaggio:

Indice disco ($), modo DOS (>) o STOP ?

Digitando il carattere $ ottieni sul video la directory del floppy collegato. Premendo RETURN ricompare il menu principale.

Digitando > lo schermo si pulisce e il carattere di prompt >, seguito da un punto interrogativo, indica che il calcolatore è in attesa di comandi DOS.
Dopo l'esecuzione di un comando RETURN fa ricomparire il menu della funzione F4.

STOP ripresenta il menu principale.

FUNZIONE F5: Videotel

Compare il Menu Videotex:

MENU VIDEOTEX

F1: Chiamata manuale	F3: Chiamata da file
F4: Modo disco	F5: Visualizza frame
F7: Editor mailbox memoria	F8: Ritorno menu principale

F1 consente di introdurre il numero telefonico necessario per collegarsi e avvia la procedura di comunicazione. Attualmente per il Videotel tale numero è 165.
Attenzione che nella comunicazione con Videotel il simbolo #va sostituito per il COMMODORE 64 con il carattere **lira.** Sul manuale del Videotel sono descritte

tutte le funzioni disponibili; tra queste risulta di particolare interesse la ricezione di programmi disponibili nella biblioteca Videotel e la stampa delle informazioni ricevute sul video. Premendo F1 si ottiene la visualizzazione della pagina di aiuto.
F3 cerca il file su disco contenente le informazioni necessarie al corretto collegamento.
F4 consente le stesse operazioni della funzione F4 già descritta (Catalog per la directory).
F5 visualizza un quadro video creato utilizzando la funzione F7 e precedentemente salvato su disco, di cui chiede il nome. Per interrompere STOP e poi RETURN per tornare al menu.
F7 consente di preparare un quadro video, di cui chiede il nome, creandolo o modificandone uno già esistente in memoria o su disco. Per conservare il quadro video devi premere F1 e poi F3.
F8 consente di tornare al menu principale; esso interrompe in qualunque momento l'operazione in corso e fa tornare al menu.

FUNZIONE F6: Selezione periferiche

Permette di selezionare il tipo di stampante collegata e l'unità di memoria di massa (cassetta o disco).

FUNZIONE F7: Terminale scrolling (P.G.E.)

Consente di utilizzare il COMMODORE 64 come terminale intelligente per collegarsi e comunicare con i servizi di Banca Dati e CBBS (Computerized Bulletin Board System) oppure con un amico che dispone di un attrezzatura analoga.
Il menu corrispondente è:

MENU TERMINALE

F1: Chiamata manuale	F3: Chiamata da file
F4: Modo disco	F5: Visualizza file
F7: Stampa file	F8: Ritorno al menu principale

F1 permette di effettuare la chiamata, o porsi in attesa di chiamata, dopo aver selezionato una tra le 10 velocità di trasmissione/ricezione visualizzate. Per chiamare le PAGINE GIALLE ELETTRONICHE puoi premere F3, infatti i parametri di trasmissione sono preselezionati. Per i collegamenti con Banche di Dati o CBBS diversi può essere necessario selezionare parametri di protollo di comunicazione diversi. Il programma contenuto nella ROM dell'adattatore telematico consente di indicare la velocità di trasmissione/ricezione, il bit di parità (mark indica che il bit di parità è posto sempre a 1, *space* indica che esso è posto sempre a 0), e se durante la

comunicazione è necessario effettuare l'eco dei caratteri digitati e il *line-feed* automatico.
F3 cerca il file su disco contenente le informazioni necessarie al corretto collegamento.
F4 consente le stesse operazioni della funzione F4 già descritta.
F5 consente di visualizzare un file di dati (anche un quadro video) precedentemente memorizzato.
F7 consente di listare un file sulla stampante.
F8 consente di tornare al menu principale; essa interrompe in qualunque momento l'operazione in corso e fa tornare al menu.

FUNZIONE F8: Ritorno al Basic

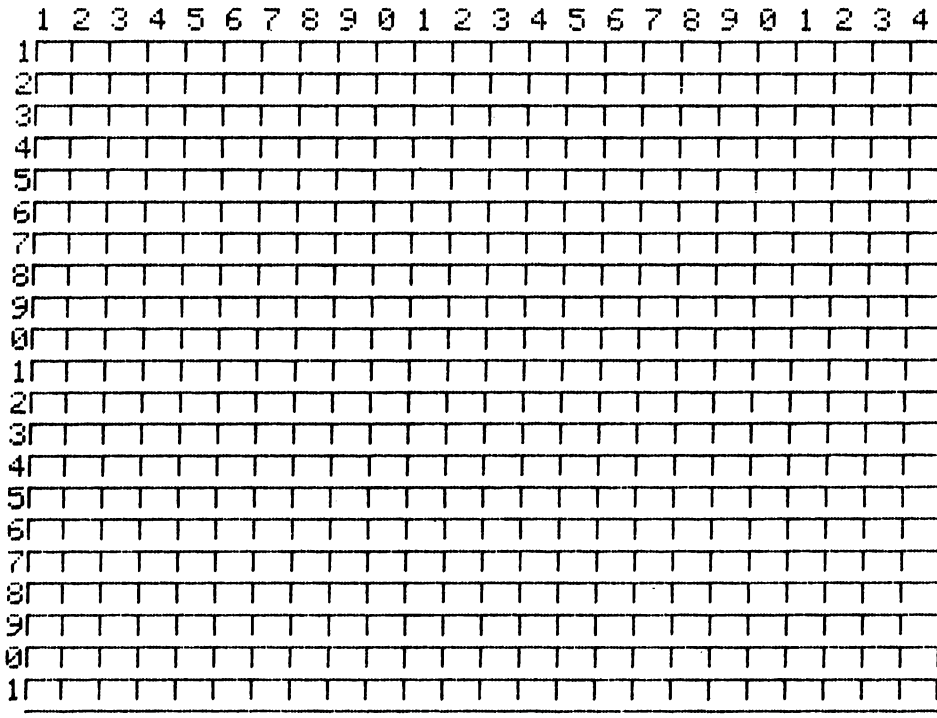
Consente di entrare in ambiente BASIC. Per attiivare nuovamente l'adattatore devi spegnere e riaccendere dopo un breve intervallo il COMMODORE 64.
Un altro modo per attivare nuovamente l'adattatore è quello di eseguire la routine in linguaggio macchina riportata nel programma CALL EDIT.

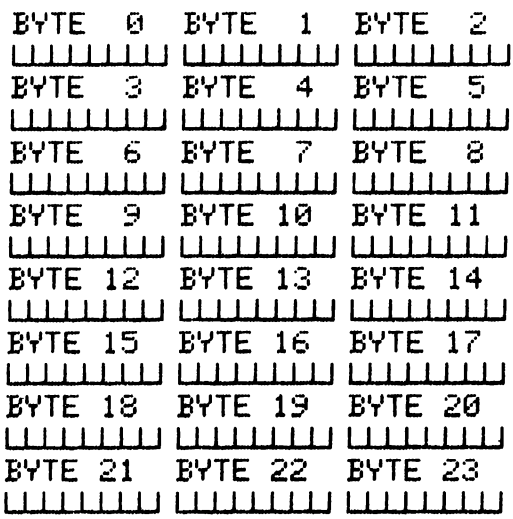
CAPITOLO 2

GLI SPRITE

2.1 COSA È UNO SPRITE

Probabilmente già sai dal manuale di istruzioni cosa sono gli SPRITE; sono dei rettangoli che puoi spostare molto facilmente, e che puoi disegnare, colorare, cambiare in maniera molto veloce e molto semplice.
In inglese SPRITE significa SPIRITO, e infatti gli sprite sono una sorta di folletti che appaiono e scompaiono con una facilità eccezionale. La loro invenzione è stata fatta dai costruttori di videogiochi, dove si sono rivelati eccezionalmente comodi ai programmatori nel realizzare le animazioni. Sono dei dispositivi abbastanza difficili da realizzare, e il COMMODORE 64 è uno dei primi calcolatori di costo modesto che li possiede.
Ma passiamo a vedere come sono fatti.
Abbiamo detto che sono dei rettangoli: la loro dimensione infatti è di 24 X 21 punti (504 punti): 24 orizzontali e 21 verticali. Possiamo quindi vedere lo sprite come una matrice di 21 X 24 punti che si può spostare attraverso lo schermo. In termine di byte la zona di memoria dedicata a uno sprite è formata da 63 byte, organizzati in 21 righe di 3 byte ciascuna.
Gli sprite si definiscono per punti in modo simile ai caratteri programmabili. Nella situazione normale, ad alta risoluzione, i punti corrispondenti ai bit 1 appaiono del colore dell'inchiostro, mentre quelli corrispondenti ai bit 0 risultano trasparenti, cioè lasciano vedere il colore di quello che sta sotto, sfondo o altro. Le immagini degli sprite possono essere memorizzate solo in blocchi di byte che inizino in un indirizzo multiplo di 64; il primo byte si riferisce agli 8 punti dell'angolo in alto a sinistra, sulla prima riga. E' possibile definire contemporaneamente 8 sprite, numerati da 0 a 7, e mostrare sul video selettivamente quello, o quelli, che di volta in volta si vuole. Per disegnare uno sprite basta prendere un foglio di carta a quadretti, riquadrare un rettangolo avente 24 quadretti di base e 21 di altezza e disegnare la sagoma voluta annerendo i quadretti. Per passare ai numeri da assegnare ai 63 byte che servono a definire il disegno in memoria, si devono considerare le 3 terne di 8 quadretti di ogni riga e trovare i numeri corrispondenti; si deve scrivere 0 per i quadretti non anneriti e 1 per quelli anneriti. La Figura 2.1 riporta uno schema di cosa serve per preparare uno sprite.


1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1


BYTE 0
BYTE 1
BYTE 2
BYTE 3
BYTE 4
BYTE 5
BYTE 6
BYTE 7
BYTE 8
BYTE 9
BYTE 10
BYTE 11
BYTE 12
BYTE 13
BYTE 14
BYTE 15
BYTE 16
BYTE 17
BYTE 18
BYTE 19
BYTE 20
BYTE 21
BYTE 22
BYTE 23

```
BYTE 24   BYTE 25   BYTE 26
└────────┘└────────┘└────────┘
BYTE 27   BYTE 28   BYTE 29
└────────┘└────────┘└────────┘
BYTE 30   BYTE 31   BYTE 32
└────────┘└────────┘└────────┘
BYTE 33   BYTE 34   BYTE 35
└────────┘└────────┘└────────┘
BYTE 36   BYTE 37   BYTE 38
└────────┘└────────┘└────────┘
BYTE 39   BYTE 40   BYTE 41
└────────┘└────────┘└────────┘
BYTE 42   BYTE 43   BYTE 44
└────────┘└────────┘└────────┘
BYTE 45   BYTE 46   BYTE 47
└────────┘└────────┘└────────┘
BYTE 48   BYTE 49   BYTE 50
└────────┘└────────┘└────────┘
BYTE 51   BYTE 52   BYTE 53
└────────┘└────────┘└────────┘
BYTE 54   BYTE 55   BYTE 56
└────────┘└────────┘└────────┘
BYTE 57   BYTE 58   BYTE 59
└────────┘└────────┘└────────┘
BYTE 60   BYTE 61   BYTE 62
└────────┘└────────┘└────────┘
```

IL DISEGNO VIENE TRACCIATO USANDO UN
CARATTERE QUALUNQUE PER ANNERIRE
LE CASELLINE DELLA MATRICE 24 X 21

I BYTE VENGONO RIEMPITI CON BIT
1 IN CORRISPONDENZA DEI CARATTERI
E COMPLETANDO CON BIT 0

Figura 2.1 Schemi per la preparazione degli SPRITE ad ALTA RISOLUZIONE

Ora, dopo aver ricordato le prime informazioni sugli sprite, proviamo il programma SPRITE1, che mostra uno sprite molto elementare sul video. Nel commento ai programmi esempio approfondiremo la conoscenza degli sprite. Ti raccomandiamo di non cambiare i numeri di linea: infatti in seguito aggiungeremo delle altre linee e costruiremo un semplice gioco, poco alla volta.

```
1 REM SPRITE1
10 REM SPRITE NERO FERMO
50 REM INDIRIZZO VIC (VIDEO INTERFACE CHIP)
51 REM (CIRCUITO GESTIONE VIDEO)
53 VI=53248
100 FORI=896 TO 958
110 POKEI,255
120 NEXTI:REM PONE 255 NELLE CELLE 896-959
130 PRINTCHR$(147)
140 REM IND. BLOCCO DI DATI/64 (896/64=14)
145 POKE2040,14
150 POKEVI,170:REM POSIZIONE ORIZZONTALE
160 POKEVI+1,120:REM POSIZIONE VERTICALE
170 POKEVI+39,0:REM COLORE NERO
180 POKEVI+21,1:REM VISUALIZZA SPRITE NUM. 0
190 PRINT"PER TOGLIERE LO SPRITE PREMI:"
195 PRINT"STOP+RESTORE"
```

Avviando il programma SPRITE1 vedi un rettangolo nero in una certa posizione dello schermo, e il cursore che lampeggia. Prova a portare il cursore vicino allo sprite usando le frecce di movimento, e a farcelo passare sopra. Ti stupisci? Il cursore non riesce a sovrapporsi allo sprite, ma gli passa sotto. Lo sprite infatti, in questo momento, ha una priorità più alta di quella dello sfondo, e quindi vi si sovrappone.
Prova a richiedere il listato del programma; vedrai che lo sfondo si sposta come sempre verso l'alto (scrolling), ma lo sprite non si muove. Anche premendo SHIFT-CLR/HOME lo sprite rimane lì; è proprio sovrapposto allo schermo. Per mandarlo via puoi premere RUN/STOP-RESTORE.
Gli sprite non si cancellano, ma si spostano.
Lo sprite numero 0 del nostro programma è memorizzato nei byte da 896 a 958. Prova a eseguire in immediato qualche POKE in uno o più dei 63 byte che lo definiscono, portandoli a 0; vedrai crearsi dei buchi nel rettangolo, attraverso i quali compare lo sfondo.

COMMENTO A SPRITE1.
Linee 50-53: gli sprites sono gestiti da un circuito integrato, il VIC II, dotato di diversi registri, che sono adiacenti e partono dal byte 53248. Assegnamo appunto a VI questo valore.
Linee 100-120: poniamo 255 nei byte che vanno da 896 a 958; sono 63 byte nei quali descriviamo lo sprite come un rettangolo pieno. Infatti 255 in binario si rappresenta come 11111111, e, in conseguenza, tutti i punti dello sprite sono ACCESI.
Linea 130: cancella lo schermo.
Linea 140: dato che abbiamo deciso di porre i dati che descrivono l'immagine dello sprite nelle locazioni che vanno da 896 in su, informiamo il VIC II che i dati relativi allo sprite partono dalla locazione 896. Di fatto le immagini degli sprites si possono porre solo in locazioni che partono da un multiplo di 64, e per questo abbiamo scelto proprio 896, che equivale a 64*14. Nel byte 2040, che è il puntatore allo sprite numero 0, poniamo quindi 14.
Linea 150: il primo registro interno del VIC II indica la X, coordinata orizzontale, dello sprite numero 0; noi poniamo lo sprite a 170 punti di distanza dall'asse delle Y, in un sistema particolare di coordinate, usato per gli sprite.
Linea 160: poniamo nel secondo registro il valore della coordinata verticale del nostro sprite (l'origine è in alto a sinistra, ed è fuori dal rettangolo visibile), posizionandolo a 120 punti di distanza dall'asse delle X.
Linea 170: il byte 53287 (40-esimo registro del VIC II) indica di che colore deve essere lo sprite numero 0 (nero in questo caso).
Linea 180: il byte 53269 (22-esimo registro del VIC II) è molto importante; esso indica quali sprites devono essere accesi (cioè mostrati sul video), e quali spenti. Il bit meno significativo (quello di posizione 0) comanda l'accensione dello sprite 0.
Linee 190-195: stampa sullo schermo il messaggio.

Vediamo ora come è organizzato il sistema di coordinate usato dal COMMODORE 64 per gli sprite. Abbiamo a disposizione un quadrante di coordinate, nel quale l'asse X si trova in alto, fuori dalla parte visibile, orientato verso destra, e l'asse delle Y si trova a sinistra, fuori dalla parte visibile, orientato verso il basso. Il video può trovarsi nelle due situazioni di 40 caratteri per 25 linee o di 38 caratteri per 24 linee; gli schemi relativi sono riportati nelle Figure 2.6 e 2.7. In ambedue le figure abbiamo segnato le coordinate dei quattro punti estremi della zona visibile dello schermo.

Affinchè uno sprite sia completamente visibile, è necessario che le coordinate X e Y del suo angolo superiore sinistro siano tali che l'intero rettangolo cada entro il video. Per uno sprite di dimensioni normali, con video 40 X 25, i limiti per X e Y sono:

$$24 <= X <= 320$$
$$50 <= Y <= 229$$

infatti 320+24=344 e 229+21=250.

Con video 38 X 24, i limiti diventano:

31 <=X <=311

54 <=Y <=225.

Gli sprite possono essere espansi nelle due direzioni raddoppiando il numero dei punti. In questo caso i limiti per le coordinate sono:

video 40 X 25

24 <=X <=296

50 <=Y <=208

video 38 X 24

31 <=X <=287

50 <=Y <=204.

Le coordinate devono essere considerate come CIRCOLARI; ciò significa che uno sprite espanso, situato come indicato nella Figura 2.8, ha per l'angolo superiore sinistro le seguenti coordinate:

X=488, infatti 488=512-24,

Y=208, infatti 208=250-42.

2.2 MOVIMENTO DI UNO SPRITE

Ora che hai visto uno sprite, probabilmente non sei soddisfatto dell'uso che ne abbiamo fatto; infatti un rettangolo nero si poteva disegnare in BASIC, con poche istruzioni POKE o addirittura con poche istruzioni PRINT. Il programma SPRITE1 non evidenzia assolutamente quello che è il maggior pregio degli sprite: la mobilità. Aggiungiamo al programma SPRITE1 le linee 55-57, 60, 150, 190-570 e cancelliamo la 160 e la 195; queste modifiche servono per far muovere il rettangolino nero atttraverso lo schermo. Per comodità riportiamo per intero il listato del programma SPRITE2, ottenuto con le modifiche.

```
1 REM SPRITE2
10 REM SPRITE CHE SI MUOVE
50 REM INDIRIZZO VIC (VIDEO INTERFACE CHIP)
51 REM (CIRCUITO GESTIONE VIDEO)
53 VI=53248
55 REM INDIRIZZO SID (SOUND INTERFACE DEVICE)
57 REM (CIRCUITO GESTIONE SUONO)
59 SI=54272
60 DX=4:DY=3:X=24:Y=50
100 FORI=896 TO 958
110 POKEI,255
```

```
120 NEXTI:REM PONE 255 NELLE CELLE 896-959
130 PRINTCHR$(147)
140 REM IND. BLOCCO DI DATI/64 (896/64=14)
145 POKE2040,14
150 POKEVI,24:POKEVI+1,50
170 POKEVI+39,0:REM COLORE NERO
180 POKEVI+21,1:REM VISUALIZZA SPRITE NUM. 0
190 HX=X/256:LX=XAND255
200 REM POSIZIONE SPRITE
205 POKEVI,LX:POKEVI+16,HX:POKEVI+1,Y
210 X=X+DX:IFX>320 OR X<24THENDX=-DX:GOSUB500
220 Y=Y+DY:IFY>229 OR Y<50 THENDY=-DY:GOSUB 500
230 GOTO190
500 REM FA BIP
510 POKESI+6,240:REM SOUSTAIN
520 POKESI+24,15:REM VOLUME
530 POKESI+1,80:REM MSB FREQUENZA
540 POKESI+4,17:REM ONDA TRIANGOLARE+ START
550 FORR=1TO3:NEXT:REM ATTESA
560 POKESI+4,16:REM ONDA TRIANGOLARE+STOP
570 RETURN
```

Il programma SPRITE2 ti mostra lo stesso rettangolo di prima, che però si muove attraverso lo schermo, e che rimbalza quando arriva ai bordi, producendo anche un piccolo suono (se il volume del televisore lo consente). Commentiamo solo le linee aggiunte.

COMMENTO A SPRITE2.
Linee 55-59: chiamiamo SI l'indirizzo del primo dei registri interni del SID, che è il circuito di gestione del suono. L'indirizzo di questo byte è 54272.
Linea 60: lo sprite si sposta se cambiamo nel tempo le sue coordinate. Noi aggiungiamo ogni volta alla X (posizione orizzontale) la variabile DX e alla Y la variabile DY. X e Y sono poste inizialmente a 24 e a 50, cioè in alto a sinistra. Il rapporto DY/DX determina l'angolo della traiettoria dello sprite. Diminuendo entrambi i valori degli incrementi delle coordinate, lo sprite si muove più lentamente, e più accuratamente (il passo è più piccolo, e lo sprite si muove poco alla volta). Aumentando DX e DY lo sprite si muove più in fretta, ma fa dei salti passando da una posizione all'altra. Se il BASIC fosse più veloce, si potrebbe lasciare piccolo il passo, e il movimento sarebbe veloce e continuo; comunque pensiamo che non sia necessario aggiungere a questo programma esempio delle routine in linguaggio

macchina, che lo renderebbero molto più veloce, dato che si riesce ad ottenere un buon risultato anche alla velocità del BASIC. Vorremmo comunque sottolineare il fatto che, in generale, il BASIC poco si adatta alla gestione della grafica, in quanto in tale campo la velocità di elaborazione è veramente essenziale per ottenere effetti brillanti.
Linea 150: assegnamo le coordinate iniziali.
Linea 190: assegnamo ora dei valori a due nuove variabili, HX e LX, che sono il byte alto e il byte basso della coordinata X. Vediamo perchè è necessario operare così. La variabile X può infatti superare il valore di 255, il massimo che possiamo mettere nel registro che determina la posizione orizzontale. I punti orizzontali di cui è composto lo schermo sono infatti 320, e noi vogliamo poter portare il nostro sprite anche oltre il 255-esimo punto. Per questa ragione esiste un registro, nel VIC II, che contiene il nono bit, il più significativo, della coordinata orizzontale di ciascuno sprite. Questo registro è il 17-esimo (numero 16, VI+16), e i suoi 8 bit sono i più significativi di ciascuna posizione X. Per lo sprite 0, di cui ci stiamo occupando, la posizione X è determinata dai seguenti 9 bit: bit 0 registro numero 16 + registro numero 0. Per lo sprite 1, invece: bit 1 registro numero 16 + registro numero 2; e così analogamente per gli altri 6 sprites.
A titolo di esempio consideriamo il seguente caso:
registro numero 16 = 00000001 binario
registro numero 0 = 00000100 binario
La posizione X dello sprite 0 in questo caso è quindi 100000100 binario, cioè 256+4, cioè 260 decimale.

Ovviamente la ragione di tali complicazioni sta nel fatto che la CPU del COMMODORE 64, la 6510, ha solo 8 bit sul bus dati, cioè ha solo 8 fili che la collegano con i vari dispositivi esterni (byte di memoria e registri), quindi, per trasmettere una informazione composta di 9 bit, occorre scrivere in due registri diversi.
Linee 200-205: X è stata ora divisa tra due variabili, HX e LX. Poniamo LX nel registro numero 0 del VIC II, e HX nel registro numero 16. LX contiene un numero da 0 a 255, cioè lo stesso numero binario di X, fatta eccezione per il nono bit, che qui è considerato sempre a zero. Se consideriamo l'esempio di prima, dove X valeva 260, LX vale 4, cioè 00000100 binario. HX, invece, tiene conto solo del nono bit, e vale 0 o 1 in accordo col valore del nono bit di X. Nel caso di prima HX valeva 1. Visto che noi visualizziamo sempre solo lo sprite 0, possiamo porre direttamente HX nel registro numero 16, in quanto i 7 bit più significativi, che noi azzeriamo sempre, non sono utilizzati. Se avessimo voluto usare lo sprite 2, anzichè lo sprite 0, anzichè HX avremmo dovuto porre HX*2↑ (numero sprite), cioè HX*4 nel caso dello sprite 2, o HX*8 nel caso dello sprite 3 (2↑3=8), e così via. Poichè 2↑0=1, è evidente che non scriviamo HX*1, in quanto moltiplicare un numero per uno è la stessa cosa che lasciarlo così com'è.
Linea 210: aggiorniamo il valore di X aggiungendo alla stessa il valore di DX. Nota

che DX viene cambiato di segno se lo sprite arriva al bordo, e viene emessa una nota per rendere più autentico l'effetto dell'urto.
Linea 220: viene fatto lo stesso per Y.
Linea 230: si torna alla linea 190, in modo da visualizzare lo sprite nella nuova posizione.
Linea 500: questa è la subroutine che viene chiamata per emettere il BIP che si sente quando lo sprite arriva al bordo.
Linea 510: ponendo 240 nel settimo registro del SID (byte 54278) poniamo SUSTAIN=15 e DECAY=0. Non scriviamo nessun numero nel registro numero 5 (ATTACK e DECAY), perchè all'accensione vi si trova già 0. Noi non facciamo quindi uso, qui, del modulatore di ampiezza che si trova nel SID. Il funzionamento del SID è abbastanza complesso, puoi andare a leggere il Capitolo 4. Se non hai mai sentito i termini ATTACK, DECAY, SUSTAIN, RELEASE, e non hai ancora visto nessun programma che usa il suono, puoi accettare queste poche linee di programma, rimandando a una fase successiva lo studio della generazione dei suoni.
Linea 520: poniamo il volume al massimo.
Linea 530: i registri 0 e 1 (primo e secondo) determinano la frequenza del suono emesso dal SID. Per essere precisi esiste questa relazione tra il numero posto nei registri della frequenza e la frequenza del suono:
F= (FH*256+FL)*FC/16777216 Hz. Dove:

- F è la frequenza del suono.
- FH è il byte più significativo della frequenza, nel nostro caso il contenuto del registro numero 1.
- FL è il byte meno significativo della frequenza, nel nostro caso il contenuto del registro numero 0, che vale 0 dall'accensione del calcolatore.
- FC è la frequenza del clock principale del calcolatore, quello che temporizza tutte le operazioni. Tale clock può avere due diversi valori, a seconda del modello del COMMODORE 64: il modello europeo ha un clock di 985248.4 Hz. La frequenza del suono che noi generiamo è quindi:

80*256*985248.4/16777216, cioè 1202.70 Hz. Tale frequenza è molto precisa e, se vuoi accordare i tuoi strumenti lo puoi fare usando il calcolatore, in quanto è assai facile calcolare la frequenza del suono emesso dal SID.
Linea 540: poniamo nel quinto registro il numero binario 00010001, avviando il generatore di suono a forma d'onda triangolare.
Linea 550: attendiamo per poco tempo, permettendo alla nota una breve durata.
Linea 560: poniamo nel quinto registro il numero binario 00010000, in modo da arrestare il generatore di suono.
Linea 570: torniamo dalla subroutine.

Ora che questo sprite quadrato si muove come se fosse una pallina, vediamo come fare per dargli anche l'aspetto di una pallina.

Riferendoci allo schema della Figura 2.2, disegnamo la pallina, poi calcoliamo i numeri binari di ogni byte e li trasformiamo in numeri decimali da inserire nel programma, per esempio per mezzo di linee DATA.
La pallina può essere disegnata come indicato nella Figura 2.2.

```
 123456789012345678901234
1                        1
2          ******        2
3        **********      3
4       ************     4
5      **************    5
6     ****************   6
7    ******************  7
8    ******************  8
9    ******************  9
0   ********************0
1   ********************1
2   ********************2
3    ******************  3
4    ******************  4
5    ******************  5
6     ****************   6
7      **************    7
8       ************     8
9        **********      9
0          ******        0
1                        1
 123456789012345678901234
```

Figura 2.2 Disegno della pallina

Ora dobbiamo trasformare in decimali tutti i 63 numeri binari che ricaviamo dal disegno della nostra pallina.
Per la prima riga è facile: tutti i punti sono spenti, quindi i primi 3 bytes valgono zero.

La seconda riga invece dà luogo a: 00000000 01111110 00000000; i dati ad essa corrispondenti in decimale sono: 0, 126, 0. Noi abbiamo calcolato tutti i 63 dati necessari per descrivere l'immagine della nostra pallina, e li abbiamo aggiunti al programma SPRITE2 sotto forma di linee DATA, ottenendo il programma SPRITE3. Inoltre abbiamo modificato le linee 100-120 nelle 100-110 per caricare i dati della pallina nello sprite numero 0.

```
1 REM SPRITE3
10 REM SPRITE CHE SI MUOVE
50 REM INDIRIZZO VIC (VIDEO INTERFACE CHIP)
51 REM (CIRCUITO GESTIONE VIDEO)
53 VI=53248
55 REM INDIRIZZO SID (SOUND INTERFACE DEVICE)
57 REM (CIRCUITO GESTIONE SUONO)
59 SI=54272
60 DX=4:DY=3:X=24:Y=50
100 REM PONE DATI PALLINA NEI BYTE 896-959
110 FORI=896TO958:READA:POKEI,A:NEXTI
130 PRINTCHR$(147)
140 REM IND. BLOCCO DI DATI/64 (896/64=14)
145 POKE2040,14
150 POKEVI,24:POKEVI+1,50
170 POKEVI+39,0:REM COLORE NERO
180 POKEVI+21,1:REM VISUALIZZA SPRITE NUM.
190 HX=X/256:LX=XAND255
200 REM POSIZIONE SPRITE PALLINA
205 POKEVI,LX:POKEVI+16,HX:POKEVI+1,Y
210 X=X+DX:IFX>320 OR X<24THENDX=-DX:GOSUB500
220 Y=Y+DY:IFY>229 OR Y<50 THENDY=-DY:GOSUB 500
230 GOTO190
500 REM FA BIP
510 POKESI+6,240:REM SOUSTAIN
520 POKESI+24,15:REM VOLUME
530 POKESI+1,80:REM MSB FREQUENZA
540 POKESI+4,17:REM ONDA TRIANGOLARE+ START
550 FORR=1TO3:NEXT:REM ATTESA
560 POKESI+4,16:REM ONDA TRIANGOLARE+STOP
570 RETURN
1000 DATA0,0,0,0,126,0,1,255,128
```

```
1010 DATA3,255,192,7,255,224,15,255,240
1020 DATA31,255,248,31,255,248,31,255,248
1030 DATA63,255,252,63,255,252,63,255,252
1040 DATA31,255,248,31,255,248,31,255,248
1050 DATA15,255,240,7,255,224,3,255,192
1060 DATA1,255,128,0,126,0,0,0,0
```

Dando il comando RUN vedi che lo sprite che si muove ora ha un aspetto nuovo; è la pallina che abbiamo disegnato prima.

COMMENTO A SPRITE3.
Linee 100-110: nella zona di memoria che abbiamo dedicato all'immagine dello sprite, poniamo i dati che leggiamo dalle linee DATA.
Linee 1000-1060: contengono i 63 dati che formano la pallina. Ogni linea DATA ha 9 dati, cioè tre righe dello sprite, e in tutto ci sono 7 linee DATA.

Ora, dopo aver fatto muovere sul video degli sprite, ti ricordiamo in cosa differisce l'animazione ottenuta con gli sprite, da quella ottenuta con la grafica normale. Nel caso della grafica normale per produrre animazione si deve:

- porre un oggetto sul video in una posizione A,
- dopo un giusto intervallo di tempo, cancellare l'oggetto dalla posizione A e porlo in una posizione B abbastanza vicina alla posizione A precedente.

Con gli sprite, invece, si visualizza l'oggetto in una posizione, poi lo si sposta, e lo spostamento lo fa sparire dalla precedente posizione.

Ora che questa pallina si muove nello schermo del tuo calcolatore, molto probabilmente ti vengono alla mente i primi videogiochi che sono apparsi nei bar, e anche in versione domestica; giochi come SQUASH, PING PONG, HOCKEY.
E' semplice, ora che c'è la pallina che si muove, aggiungere una racchetta e fare il gioco dello SQUASH.

Disegnamo la nostra racchetta come in Figura 2.3.

```
 123456789012345678901234
1         ********        1
2         ********        2
3         ********        3
4         ********        4
5         ********        5
6         ********        6
7         ********        7
8         ********        8
9         ********        9
0         ********        0
1         ********        1
2         ********        2
3         ********        3
4         ********        4
5         ********        5
6         ********        6
7         ********        7
8         ********        8
9         ********        9
0         ********        0
1         ********        1
 123456789012345678901234
```

Figura 2.3 Disegno della racchetta

I dati di questa racchetta sono davvero semplici: in binario ogni linea è così: 00000000 11111111 00000000, cioè 0, 255, 0 in decimale. Possiamo modificare il programma SPRITE3, ottenendo SPRITE4, con due oggetti sul video, usando anche lo sprite numero 1.

```
1 REM SPRITE4
10 REM SPRITE CHE SI MUOVE
50 REM INDIRIZZO VIC (VIDEO INTERFACE CHIP)
51 REM (CIRCUITO GESTIONE VIDEO)
53 VI=53248
55 REM INDIRIZZO SID (SOUND INTERFACE DEVICE)
```

```
57 REM (CIRCUITO GESTIONE SUONO)
59 SI=54272
60 DX=4:DY=3:X=24:Y=50
70 JA=56320:REM INDIRIZZO PORTA JOYSTICK
100 REM PONE DATI PALLINA NEI BYTE 896-959
110 FORI=896TO958:READA:POKEI,A:NEXTI
125 FORI=960TO1022STEP3:POKEI,0:POKEI+1,255
126 POKEI+2,0:NEXT:REM DATI RACCHETTA
130 PRINTCHR$(147)
140 REM IND. BLOCCO DI DATI/64 (896/64=14)
143 POKE2040,14
145 REM RACCHETTA:IND.BLOCCO DATI/64 (960/64=15)
147 POKE2041,15
150 POKEVI,24:POKEVI+1,50
160 POKEVI+2,255
170 POKEVI+39,0:REM COLORE NERO
171 POKEVI+40,1:REM RACCHETTA COLORE BIANCO
180 POKEVI+21,3:REM VISUALIZZA SPRITE NUM. 0,1
190 HX=X/256:LX=XAND255
200 REM POSIZIONE SPRITE PALLINA
202 POKEVI,LX:POKEVI+16,HX:POKEVI+1,Y
204 IFPEEK(JA)AND1THENAY=AY+4:IFAY>229THENAY=229
205 IFPEEK(JA)AND2THENAY=AY-4:IFAY<50THENAY=50
208 POKEVI+3,AY
210 X=X+DX:IFX>320 OR X<24THENDX=-DX:GOSUB500
220 Y=Y+DY:IFY>229 OR Y<50 THENDY=-DY:GOSUB 500
230 GOTO190
500 REM FA BIP
510 POKESI+6,240:REM SOUSTAIN
520 POKESI+24,15:REM VOLUME
530 POKESI+1,80:REM MSB FREQUENZA
540 POKESI+4,17:REM ONDA TRIANGOLARE+ START
550 FORR=1TO3:NEXT:REM ATTESA
560 POKESI+4,16:REM ONDA TRIANGOLARE+STOP
570 RETURN
1000 DATA0,0,0,0,126,0,1,255,128
1010 DATA3,255,192,7,255,224,15,255,240
1020 DATA31,255,248,31,255,248,31,255,248
1030 DATA63,255,252,63,255,252,63,255,252
1040 DATA31,255,248,31,255,248,31,255,248
1050 DATA15,255,240,7,255,224,3,255,192
1060 DATA1,255,128,0,126,0,0,0,0
```

In SPRITE4 vi sono due oggetti sullo schermo: la pallina, che si muove da sola, e la racchetta, che si muove con il joystick inserito nella porta B. Ti sei sicuramente accorto che la racchetta non colpisce la pallina; non abbiamo ancora aggiunto quella parte di programma, la aggiungeremo alla prossima modifica.

COMMENTO A SPRITE4.
Linea 70: inizializziamo una nuova variabile, JA, che contiene l'indirizzo in cui leggere lo stato del joystick.
Linee 125-126: in questo ciclo poniamo in una zona di memoria i dati che rappresentano l'immagine della racchetta, nei byte che partono dall'indirizzo 960 (15-esimo blocco di 64 bytes).
Linee 145-147: il puntatore dei dati dello sprite 1 viene posto a 15, indicando al VIC II che i dati con l'immagine dello sprite 1 partono dalla locazione di memoria 960.

Linea 160: poichè la racchetta si deve muovere verticalmente, poniamo il valore della sua coordinata X nell'apposito registro del VIC II, cambieremo in seguito solo la sua coordinata Y. Poniamo la X della racchetta a 255.
Linea 171: Indichiamo al VIC II il colore dello sprite numero 1 ponendo il codice nel 41-esimo registro.
Linea 180: a differenza di prima, quando mostravamo un solo sprite, ora ne mostriamo due, e per fare questo poniamo nel registro per l'abilitazione degli sprites il numero 3, che in binario è 00000011; come si vede subito in questo numero vi sono due bit a 1, i bit di posizione 0 e 1. Il VIC II mostrerà adesso lo sprite 0 e lo sprite 1.
Linee 204-205: in queste linee viene letto lo stato del joystick; se si va in basso, si incrementa AY, se si va in alto la si decrementa. La variabile AY contiene la posizione della racchetta: se non vuoi usare il joystick, ma la tastiera, puoi facilmente cambiare queste linee: GET A$: IF A$=... ecc...; se vuoi usare i paddle, adatta tu il programma, sapendo che i valori dei due paddle (numeri da 0 a 255) si trovano negli indirizzi SI+25 e SI+26. Puoi direttamente porre : AY=PEEK(SI+25).
Linea 208: poniamo nel registro 3 del VIC II il valore di AY; il registro 3 indica la posizione verticale dello sprite 1 (quello che usiamo per la racchetta).

E' già stata annunciata la prossima modifica; con l'aggiunta di qualche linea, ora scopriremo l'eventuale collisione tra pallina e racchetta. Aggiungi queste linee:

```
223 CL=PEEK(VI+30)
224 IFCL=3 THEN IF DX > 0 THEN DX=-DX: GOSUB 500
```

per ottenere il programma SPRITE5, del quale non riportiamo il listato, ma lo trovi registrato sulla cassetta.

Avvia l'esecuzione del programma, e ti accorgi che ora la racchetta funziona. Come probabilmente hai già osservato, non abbiamo fatto riferimento alle posizioni di racchetta e pallina; facciamo riferimento solamente a un registro del VIC II, il 31-esimo registro, ed assegnamo alla variabile CL il valore di questo registro. CL sta per COLLISIONE. Il 31-esimo registro del VIC II infatti ci indica se c'è stata collisione tra gli sprite. Se la collisione c'è stata, i bit corrispondenti agli sprite interessati dalla collisione sono a 1. Se quindi si sono scontrati lo sprite 0 (la pallina) con lo sprite 1 (la racchetta) noi troviamo a 1 il bit 0 e il bit 1: troviamo cioè il numero binario 00000011 nel registro collisione sprite-contro sprite (31-esimo registro, il numero 30). Come vedi la pallina si sovrappone alla racchetta al momento dell'impatto; questo dipende dal fatto che la priorità della pallina è superiore a quella della racchetta. Le priorità degli sprite dipendono dal loro numero; lo sprite numero 0 ha priorità 0, la più alta, lo sprite numero 7 ha priorità 7, la più bassa. Nella Figura 2.4 sono indicate le priorità degli sprite.

Linea 223: pone nella variabile CL il valore del registro collisioni sprite-sprite.
Linea 224: se c'è collisione cambia la direzione del moto orizzontale della pallina, ed emette il bip, a patto che la pallina vada da sinistra verso destra (DX > 0).

La priorità tra gli sprite e lo sfondo è controllata dal registro numero 27 (il 28-esimo, di indirizzo 53275); quando il bit relativo a uno sprite è a 0, lo sprite ha priorità maggiore dello sfondo, quando è a 1 lo sfondo prevale sullo sprite. La differenza di priorità fra gli sprite fa sì che un oggetto appaia davanti all'altro e produce un effetto tridimensionale. Se nel programma SPRITE1, dopo il suo arresto, scendi con il cursore e esegui in immediato POKE 53275,1, e dopo sposti il cursore e lo fai passare attraverso lo sprite, questa volta il cursore passa sopra. Infatti per effetto della POKE eseguita, la priorità dello sfondo è diventata superiore a quella dello sprite numero 0.

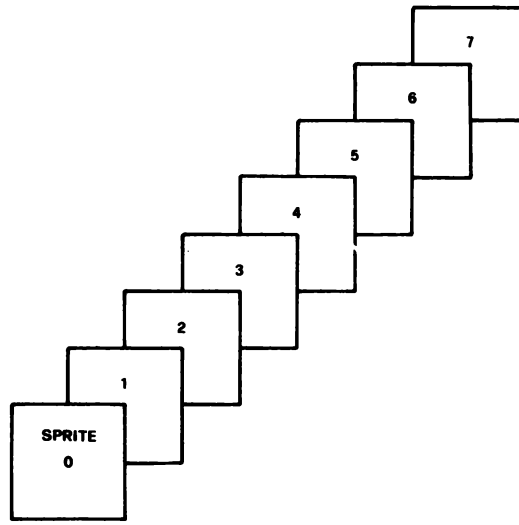


Figura 2.4 Priorità tra gli Sprite

2.3 ALTRI ESEMPI DI ANIMAZIONE

Ora che abbiamo completato il gioco dello SQUASH (SPRITE5), possiamo andare avanti e aggiungere altre modifiche per ottenere il gioco del PING PONG (SPRITE6).
Riportiamo l'intero listato del programma SPRITE6.

```
1 REM SPRITE6
10 REM GIOCO PING PONG
20 REM COLORE QUADRO BLU+SFONDO GRIGIO
25 POKE53280,12:POKE53281,6
50 REM INDIRIZZO VIC (VIDEO INTERFACE CHIP)
51 REM (CIRCUITO GESTIONE VIDEO)
```

```
53 VI=53248
55 REM INDIRIZZO SID (SOUND INTERFACE DEVICE)
57 REM (CIRCUITO GESTIONE SUONO)
59 SI=54272
60 DX=9:DY=6  :X=24:Y=50
70 REM INDIRIZZI PORTE JOYSTICK
75 JA=56320:JB=56321
80 AY=140:BY=AY
100 FORI=896 TO 957
110 READA:POKEI,A
120 NEXTI:REM PONE DATI PALLINA IN MEMORIA
125 FORI=960TO1022STEP3:POKEI,0:POKEI+1,255
126 POKEI+2,0:NEXT:REM DATI RACCHETTE
130 PRINTCHR$(147)
140 REM IND.BLOCCO DATI/64 (896/64=14)
141 POKE2040,14
142 REM RACCHETTA:IND.BLOCCO DATI/64 (960/64=15)
143 POKE2041,15
144 REM IND.BLOCCO DI DATI/64 (958/64=15)
145 POKE2042,15
150 POKEVI,24:POKEVI+1,50:REM POSIZ. INIZIALE
155 POKEVI+2,16:POKEVI+4,72
160 POKEVI+1,120:REM POSIZIONE VERTICALE
170 POKEVI+39,12:REM COLORE PALLINA: GRIGIO
171 POKEVI+40,1:REM COLORE RACCHETTA A: BIANCO
172 POKEVI+41,0:REM COLORE RACCHETTA B: NERO
180 REM VISUALIZZA SPRITES NUM. 0,1,2
185 POKEVI+21,7
190 HX=X/256:LX=XAND255
200 REM POSIZIONE SPRITE
202 POKEVI,LX:POKEVI+16,HX+4:POKEVI+1,Y
204 IFPEEK(JA)AND1THENAY=AY+4:IFAY>229THENAY=229
205 IFPEEK(JA)AND2THENAY=AY-4:IFAY<50THENAY=50
206 IFPEEK(JB)AND1THENBY=BY+4:IFBY>229THENBY=229
207 IFPEEK(JB)AND2THENBY=BY-4:IFBY<50THENBY=50
208 POKEVI+3,AY:POKEVI+5,BY
210 X=X+DX:IFX>345 OR X<1THENDX=-DX:GOSUB600
220 Y=Y+DY:IFY>229 OR Y<50 THENDY=-DY:GOSUB 500
223 CL=PEEK(VI+30)
224 IFCL=3THENIFDX<0THENDX=-DX:GOSUB500
226 IFCL=5THENIFDX>0THENDX=-DX:GOSUB500
```

```
230 GOTO190
500 REM FA BIP
510 POKESI+6,240:REM SOUSTAIN
520 POKESI+24,15:REM VOLUME
530 POKESI+1,80:REM MSB FREQUENZA
540 POKESI+4,17:REM ONDA TRIANGOLARE+START
550 FORR=1TO3:NEXT:REM ATTESA
560 POKESI+4,16:REM ONDA TRIANGOLARE+STOP
570 RETURN
600 REM FA BIIIIP
610 POKESI+4,17:REM ONDA TRIANGOLARE+START
620 FORR=1TO1000:NEXT:REM ATTESA
630 POKESI+4,16:REM ONDA TRIANGOLARE+STOP
640 PB=PB-(DX>0):PA=PA-(DX<0)
650 PRINTCHR$(147)TAB(16)CHR$(5)PACHR$(144)PB
660 RETURN
1000 DATA0,0,0,0,0,0,0,126,0,1,255,128
1010 DATA3,255,192,7,255,224,15,255,240
1020 DATA31,255,248,31,255,248,31,255,248
1030 DATA63,255,252
1040 DATA31,255,248,31,255,248,31,255,248
1050 DATA15,255,240,7,255,224,3,255,192
1060 DATA1,255,128,0,126,0,0,0,0,0,0,0
```

A questo punto nel gioco sono presenti due racchette di colore diverso, comandate dai due joystick, e una pallina; adoperiamo quindi 2 blocchi di dati (pallina e racchetta), ma la racchetta compare in due posizioni diverse con colore diverso. Se vuoi puoi sostituire i joystick con i paddle, ma devi modificare la linea 208 come segue:

208 POKE VI+3, PEEK(SI+25): POKE VI+5, PEEK(SI+26)

Nel commento mettiamo in evidenza solo le novità rispetto ai precedenti programmi esempio.

COMMENTO A SPRITE6.
Linea 20: cambiano i colori dello schermo.
Linea 60: cambiano il passo orizzontale DX e quello verticale DY. Il programma deve ora gestire due racchette, e quindi il tempo che passa tra uno spostamento della pallina e l'altro è più lungo; per questa ragione la pallina compie dei passi più grandi, per non muoversi troppo lentamente.
Linea 70: definiamo anche JB, che è l'indirizzo della porta di I/O dove è collegato il secondo joystick.
Linea 144: inizializziamo il puntatore del blocco dei dati della seconda racchetta.
Linea 155: poniamo nei registri del VIC II le posizioni orizzontali delle racchette.
Linee 160-185: sono spiegate dai commenti di linea.
Linea 190: calcola la parte più significativa e quella meno significativa di X.
Linea 200: posiziona la pallina.
Linee 206-207: legge la posizione del secondo joystick.
Linea 208: aggiorna la posizione delle racchette.
Linea 210: aggiorna la posizione orizzontale della pallina e fa BIIIIP se arriva al bordo senza essere colpita dalla racchetta.
Linea 600: sottoprogramma che genera un suono lungo e aggiorna il punteggio della partita. Funziona come il sottoprogramma in 500.
Linea 640: forse questa linea ti sembra un pò strana. la variabile BP contiene il punteggio del secondo giocatore; essa viene incrementata solo se la pallina andava verso destra (DX > 0), in tale caso l'espressione condizionale (DX > 0) è verificata, e quindi per il calcolatore essa dà luogo a un numero di valore −1. Quando, invece, l'espressione condizionale non viene verificata, questa assume il valore 0.
Linea 650: stampa il punteggio.

Come vedi non sono stati aggiunti concetti nuovi; abbiamo solo vivacizzato un pò il gioco precedente. Puoi provare a cambiare qualche parametro, come la velocità della pallina o delle racchette, per acquistare maggiore dimestichezza con l'argomento dell'animazione con gli sprite.

Ti suggeriamo ora di modificare alcune linee del programma SPRITE6, per ottenere il programma SPRITE7, realizzando così il gioco HOKEY.
Abbiamo aggiunto alcune linee, che sono riportate come MOD-SPRITE6, al programma precedente (SPRITE6), che commentiamo. Non riportiamo il listato completo del programma SPRITE7, che è contenuto sulla cassetta.

```
0 REM MOD-SPRITE6
1 REM SPRITE7
10 REM GIOCO HOKEY
146 REM RACCHETTA:IND.BLOCCO DATI/64 (960/64=15)
147 POKE2043,15
148 REM RACCHETTA:IND.BLOCCO DATI/64 (960/64=15)
149 POKE2044,15
155 REM POSIZ. VERT. RACCHETTE
157 POKEVI+2,16:POKEVI+4,72
159 POKEVI+6,255:POKEVI+8,89
173 POKEVI+42,1:REM COLORE RACCHETTA A: BIANCO
174 POKEVI+43,0:REM COLORE RACCHETTA B: NERO
180 REM VISUALIZZA SPRITES NUM. 0,1,2,3,4
185 POKEVI+21,31
209 POKEVI+7,AY:POKEVI+9,BY
225 IFCL=9THENGOSUB700
227 IFCL=17THENGOSUB800
700 REM COLLISIONE PALLINA-ATTACCANTE A
710 IFDX<0THENDX=-DX:GOSUB500
720 RETURN
800 REM COLLISIONE PALLINA-ATTACCANTE B
810 IFDX>0THENDX=-DX:GOSUB500
820 RETURN
```

COMMENTO A SPRITE7.
Linea 10: cambiato il nome.
Linee 146-149: indirizzo blocco dati seconda racchetta.
Linee 155-159: posizione racchette.
Linea 173: colore prima racchetta.
Linea 174: colore seconda racchetta.
Linee 180-185: visualizza gli sprite.
Linea 209: aggiorna posizione racchetta.
Linee 225 e 227: controlli posizione.
Linee 700-720: collisione pallina-prima racchetta.
Linee 800-820: collisione pallina-seconda racchetta.

Abbiamo preso in esame tutti gli argomenti che è necessario conoscere per produrre animazione con gli sprite. Ora ti presentiamo una serie di programmi, ottenuti uno dall'altro con modifiche successive. In questi programmi ti mostriamo come sia possibile evitare noiosi calcoli per programmare uno sprite.

Cominciamo con il programma SPRITE8; in esso prepariamo la sagoma di un omino in piedi, usando la tecnica di disegnarlo con asterischi e spazi in 21 linee DATA. Provvede il programma a trasformare i contenuti delle linee DATA nei valori da memorizzare nei 63 byte per programmare lo sprite.

```
1 REM SPRITE8
10 REM OMINO FERMO
20 POKE56,32:CLR:POKE53280,0:POKE53281,0
30 DIMD(63)
40 DIMZ(7):FORI=0TO7:Z(I)=2↑I:NEXT
50 VI=53248:REM INDIRIZZO DEL VIC
100 GOSUB2000
110 ND=128:GOSUB2100
130 PRINTCHR$(147)
140 POKE2040,128:REM IND. BLOCCO DI DATI/64
150 POKEVI,170:REM POSIZIONE ORIZZONTALE
160 POKEVI+1,120:REM POSIZIONE VERTICALE
170 POKEVI+39,7:REM COLORE NERO
180 POKEVI+21,1:REM VISUALIZZA SPRITE NUM. 0
190 PRINT"PER TOGLIERE LO SPRITE PREMI:"
195 PRINT"STOP+RESTORE"
200 END
999 REM   123456789012345678901234
1000 DATA"          *****         "
1010 DATA"         **** *         "
1020 DATA"         *******        "
1030 DATA"         ******         "
1040 DATA"          ****          "
1050 DATA"           **           "
1060 DATA"          ****          "
1070 DATA"         **  **         "
1080 DATA"        ***  **         "
1090 DATA"        ***  ***        "
1100 DATA"        ***  ***        "
1110 DATA"        ***  ***        "
1120 DATA"        ****  **        "
1130 DATA"         ******         "
1140 DATA"          ****          "
1150 DATA"           **           "
1160 DATA"           **           "
```

```
1170 DATA"            **            "
1180 DATA"            **            "
1190 DATA"            ****          "
1200 DATA"            ****          "
1210 REM  123456789012345678901234
2000 PRINTCHR$(147);:FORI=0TO20:READA$:PRINTA$
2010 FORJ=0TO2:D(I*3+J)=0:B$=MID$(A$,1+J*8,8)
2020 FORK=1TO8
2025 IFMID$(B$,K,1)="*"THEN2035
2030 NEXTK,J,I:RETURN
2035 D(I*3+J)=D(I*3+J)+Z(8-K):GOTO2030
2100 FORI=0TO63:POKEND*64+I,D(I):NEXT:RETURN
```

COMMENTO A SPRITE8.
Linea 20: pone 32 nel byte di indirizzo 56; questo produce l'effetto di abbassare il top della memoria, lasciando le locazioni di indirizzo superiore a 8192 (32*256=8192) a nostra disposizione, senza che il BASIC possa accedervi.
Linee 30-40: dimensiona il vettore D che conterrà i valori numerici dello sprite e prepara una matrice Z di 8 elementi con le prime 8 potenze del 2 (risulta più veloce accedere ad un elemento di una matrice che calcolare una potenza).
Linee 100-200: prepara i parametri necessari, come indicato dal commento contenuto nelle REM.
Linee 1000-1200: linee DATA con l'immaginé grafica dello sprite, disegnata con gli asterischi.
Linee 2030-2035: sottoprogramma che pone in memoria i dati della matrice D a partire dall'indirizzo ND*64. ND=128 significa che i dati sono memorizzati a partire dalla locazione 8192 (128*64=8192), che è la prima di quelle riservate con la linea di programma 20. Mentre vengono calcolati i dati, il programma mostra gli sprite ingranditi sul video.

Con SPRITE8 otteniamo un omino fermo; le dimensioni dell'omino sono quelle possibili per una sagoma contenuta in un rettangolo di 24 X 21 punti. E' possibile espandere lo sprite nelle due direzioni; si può:

- raddoppiare la larghezza,
- raddoppiare l'altezza,
- raddoppiare sia la larghezza che l'altezza.

Per provare anche questa caratteristica degli sprite, puoi aggiungere al programma SPRITE8 le due linee seguenti:

```
163 POKE VI+23,1:REM ESPANSIONE VERTICALE SPRITE 0
166 POKE VI+29,1:REM ESPANSIONE ORIZZONTALE SPRITE 0
```

e provare il programma. Puoi anche provare cosa succede aggiungendo una sola delle due linee suggerite.

Ora aggiungiamo poche linee a SPRITE8 (nella versione precedente alle modifiche suggerite per ingrandire l'omino) per ottenere il programma SPRITE9, che non listiamo, ma trovi sulla cassetta. In esso il nostro omino scivola sul video, senza muovere le gambe. Le linee aggiunte sono riportate come MOD-SPRITE8.

```
0 REM MOD-SPRITE8
1 REM SPRITE9
10 REM OMINO CHE SI MUOVE
200 FORX=0TO340
210 HX=X/256:LX=XAND255
220 POKEVI,LX:POKEVI+16,HX
230 NEXT:GOTO200
```

Ora vediamo di migliorare ancora la situazione, cioè facciamo camminare l'omino. Per ottenere il programma SPRITE10, memorizzato sulla cassetta, ma non listato qui per intero, abbiamo aggiunto le linee riportate come MOD-SPRITE9. Queste linee disegnano da 1210 a 1410 un omino in posizione di passo. Le altre linee aggiunte servono per alternare le due immagini (i due sprite) sullo schermo e dare l'impressione del movimento. Le linee 210-215 realizzano il movimento così: la variabile S contiene 0 o 1, a seconda del valore del bit di posizione 2 della variabile X; ogni 4 spostamenti S cambia valore, determinando quale delle due immagini deve essere visualizzata.

```
0 REM MOD-SPRITE9
1 REM SPRITE10
10 REM OMINO CHE CAMMINA
120 GOSUB2000
125 ND=129:GOSUB2100
210 HX=X/256:LX=XAND255:S=-((XAND4)=0)
215 POKE2040,128+S
220 POKEVI,LX+S:POKEVI+16,HX
```

```
1210 DATA"          *****          "
1220 DATA"         **** *          "
1230 DATA"         *******         "
1240 DATA"         ******          "
1250 DATA"          ****           "
1260 DATA"           **            "
1270 DATA"          ****           "
1280 DATA"         **  **          "
1290 DATA"        ***  **          "
1300 DATA"        ****  **         "
1310 DATA"        ****  **         "
1320 DATA"        *****  *         "
1330 DATA"        ******           "
1340 DATA"         ******          "
1350 DATA"          ****           "
1360 DATA"          *****          "
1370 DATA"         **  **          "
1380 DATA"         **  **          "
1390 DATA"         **   **         "
1400 DATA"        ***   ***        "
1410 DATA"        ****  ****       "
```

Per concludere ti presentiamo un semplice gioco per due persone. Esso, realizzato nel programma SPRITE11, deriva dai 3 programmi degli omini, appena presentati. Il gioco consiste nel far mangiare al proprio omino il maggior numero possibile di palline. Per muovere l'omino, ogni giocatore usa un joystick; le palline vengono mangiate urtandole con la pancia. Per bloccare l'avversario si può usare il bottone del fuoco in qualunque momento, l'avversario sviene per un breve periodo, ma tu perdi 5 palline. Le palline vengono distribuite sul video in modo casuale. Nella parte alta del video viene visualizzato il punteggio.

Il programma SPRITE11 non presenta alcuna sostanziale differenza rispetto ai precedenti, per cui ne riportiamo il listato senza commenti. Come vedi abbiamo dovuto disegnare 5 sprite diversi.

```
1 REM SPRITE11
10 REM GIOCHINO DUE OMINI AVVERSARI
110 POKE56,32:CLR:POKE53280,0:POKE53281,0
120 DIMD(63):VI=13*4096:CP=81
125 DIMZ(7):FORI=0TO7:Z(I)=2↑I:NEXT
130 GOSUB1530:ND=128:GOSUB1610
140 GOSUB1530:ND=129:GOSUB1610
150 GOSUB1530:ND=130:GOSUB1610
160 GOSUB1530:ND=131:GOSUB1610
170 GOSUB1530:ND=134:GOSUB1610
180 RESTORE:GOSUB1570:ND=132:GOSUB1610
190 GOSUB1570:ND=133:GOSUB1610
200 POKE2040,130:POKE2041,130:X=128:Y=128
205 W=200:Z=200:POKEVI+21,3
210 POKEVI+39,3:POKEVI+40,2
220 A=RND(-1)
230 POKE53281,7:PRINT"□":POKE53281,0
235 FORI=1TO50:POKE1024+RND(1)*1000,CP:NEXT
240 POKEVI,XAND255:POKEVI+2,WAND255
245 POKEVI+16,((XAND256)/256)OR((WAND256)/128)
250 POKEVI+1,Y:POKEVI+3,Z
260 IFM1THENPOKE2040,134:GOTO280
265 XR=((DIAND2)=0)*(-(XAND4)/4)
266 XS=((DIAND2)=2)*(-(YAND4)/4)
267 XR=XR OR XS OR 128 OR DI
270 POKE2040,XR
280 IFM2THENPOKE2041,134:GOTO300
290 XR=((D2AND2)=0)*(-(WAND4)/4)
293 XS=((D2AND2)=2)*(-(ZAND4)/4)
295 POKE2041,XR OR XS OR 128 OR D2
300 J=NOTPEEK(56320):K=NOTPEEK(56321)
310 IFM1THENM1=M1-1:J=0:K=KAND15
320 IFM2THENM2=M2-1:K=0:J=JAND15
330 IFJAND1THENY=Y-4:Y=YAND255:DI=2
340 IFJAND2THENY=Y+4:Y=YAND255:DI=2
350 IFJAND4THENX=X-4:X=XAND511:DI=4
360 IFJAND8THENX=X+4:X=XAND511:DI=0
370 IFJAND16THENIFP1>4THENP1=P1-5:M2=50
380 IFKAND1THENZ=Z-4:Z=ZAND255:D2=2
390 IFKAND2THENZ=Z+4:Z=ZAND255:D2=2
400 IFKAND4THENW=W-4:W=WAND511:D2=4
```

```
410 IFKAND8THENW=W+4:W=WAND511:D2=0
420 IFKAND16THENIFP2>4THENP2=P2-5:M1=50
430 C=PEEK(VI+31)
440 IFCAND1THEN473
450 IFCAND2THEN476
460 PRINT"▚"P1"▌    ▞",P2"▌    "
470 GOTO240
473 A=1024+(X-16)/8+INT((Y-40)/8)*40
474 IFPEEK(A)=CPTHENPOKEA,32:P1=P1+1
475 GOTO450
476 A=1024+(W-16)/8+INT((Z-40)/8)*40
477 IFPEEK(A)=CPTHENPOKEA,32:P2=P2+1
478 GOTO460
479 REM  123456789012345678901234
480 DATA"          *****         "
490 DATA"         **** *         "
500 DATA"         *******        "
510 DATA"         ******         "
520 DATA"          ****          "
530 DATA"           **           "
540 DATA"          ****          "
550 DATA"         **  **         "
560 DATA"        ***  **         "
570 DATA"        ***  ***        "
580 DATA"        ***  ***        "
590 DATA"        ***  ***        "
600 DATA"        ****  **        "
610 DATA"         ******         "
620 DATA"          ****          "
630 DATA"           **           "
640 DATA"           **           "
650 DATA"           **           "
660 DATA"           **           "
670 DATA"           ****         "
680 DATA"           ****         "
690 DATA"          *****         "
700 DATA"         **** *         "
710 DATA"         *******        "
720 DATA"         ******         "
730 DATA"          ****          "
740 DATA"           **           "
750 DATA"          ****          "
```

```
760 DATA"          **  **          "
770 DATA"         ***  **          "
780 DATA"         ****  **         "
790 DATA"         ****  **         "
800 DATA"         *****  *         "
810 DATA"         ******           "
820 DATA"          ******          "
830 DATA"           ****           "
840 DATA"           *****          "
850 DATA"          **  **          "
860 DATA"          **  **          "
870 DATA"          **   **         "
880 DATA"         ***   ***        "
890 DATA"         ****  ****       "
900 DATA"           *****          "
910 DATA"          ******* **      "
920 DATA"          ******* **      "
930 DATA"           *****  **      "
940 DATA"       **   ***   **      "
950 DATA"       **   ***   **      "
960 DATA"       *** ***** ***      "
970 DATA"        * ******* *       "
980 DATA"         * ***** *        "
990 DATA"         ** *** **        "
1000 DATA"         *** * ***        "
1010 DATA"         **** ****        "
1020 DATA"          ** * **         "
1030 DATA"          * *** *         "
1040 DATA"          * *** *         "
1050 DATA"           *****          "
1060 DATA"           ** **          "
1070 DATA"           ** **          "
1080 DATA"           **             "
1090 DATA"           **             "
1100 DATA"           **             "
1110 DATA"           *****          "
1120 DATA"        ** *******        "
1130 DATA"        ** *******        "
```

```
1140 DATA"       **  *****          "
1150 DATA"       **   ***   **      "
1160 DATA"       **   ***   **      "
1170 DATA"       *** ***** ***      "
1180 DATA"        * ******* *       "
1190 DATA"         * ***** *        "
1200 DATA"         ** *** **        "
1210 DATA"         *** * ***        "
1220 DATA"         **** ****        "
1230 DATA"          ** * **         "
1240 DATA"          * *** *         "
1250 DATA"          * *** *         "
1260 DATA"           *****          "
1270 DATA"           ** **          "
1280 DATA"           ** **          "
1290 DATA"               **         "
1300 DATA"               **         "
1310 DATA"               **         "
1320 DATA"                          "
1330 DATA"                          "
1340 DATA"                          "
1350 DATA"                          "
1360 DATA"                          "
1370 DATA"                          "
1380 DATA"                          "
1390 DATA" ** **                    "
1400 DATA"                          "
1410 DATA"**   **                   "
1420 DATA"   **                     "
1430 DATA"                          "
1440 DATA"                          "
1450 DATA"   *     ***  *           "
1460 DATA" ****  ***   **           "
1470 DATA"* *** ***   ****          "
1480 DATA"*******    *****      ** "
1490 DATA"*******  *******      ** "
1500 DATA"***** ****************** "
1510 DATA" ***   **************** "
1520 DATA"         *************** "
1525 REM  12345678901234567890123 4
1530 PRINT"□";:FORI=0TO20:READA$:PRINTA$
1540 FORJ=0TO2:D(I*3+J)=0:B$=MID$(A$,1+J*8,8)
```

```
1550 FORK=1TO8
1555 IFMID$(B$,K,1)="*"THEND(I*3+J)=D(I*3+J)+Z(8-K)
1560 NEXTK,J,I:RETURN
1570 PRINT"□";:FORI=0TO20:READA$:PRINTA$
1580 FORJ=0TO2:D(I*3+J)=0:B$=MID$(A$,1+(2-J)*8,8)
1585 FORJ=0TO2:D(I*3+J)=0:B$=MID$(A$,1+(2-J)*8,8)
1590 FORK=1TO8
1595 IFMID$(B$,K,1)="*"THEND(I*3+J)=D(I*3+J)+Z(K-1)
1600 NEXTK,J,I:RETURN
1610 FORI=0TO63:POKEND*64+I,D(I):NEXT:RETURN
```

2.4 SPRITE MULTICOLORE

In tutti i programmi esempio riportati nei precedenti paragrafi abbiamo usato gli sprite nel modo alta risoluzione. Nel capitolo dedicato alla grafica abbiamo visto la differenza tra il modo ALTA RISOLUZIONE e il modo MULTICOLORE. Nel primo ad ogni singolo bit corrisponde un punto, nel secondo un punto corrisponde a due bit, ragione per la quale il punto può essere di uno tra 4 colori diversi, a seconda del valore dei 2 bit che lo definiscono. Per valore 00 abbiamo il colore dello sfondo, per 01 il colore ausiliario numero 0 (registro numero 37), per 10 il colore dello sprite, per 11 il colore ausiliario numero 1 (registro numero 38).

In questo modo i pixel a disposizione per disegnare uno sprite diminuiscono; la matrice diventa 12 X 21. Ogni pixel risulta largo 2 punti ad alta risoluzione e può essere di uno tra 4 colori scelti tra i 16 disponibili.

Nella Figura 2.5 riportiamo gli schemi necessari per disegnare uno sprite multicolore.

	1	2	3	4	5	6	7	8	9	10	11	12
1												
2												
3												
4												
5												
6												
7												
8												
9												
0												
1												
2												
3												
4												
5												
6												
7												
8												
9												
0												
1												

BYTE 0	BYTE 1	BYTE 2
BYTE 3	BYTE 4	BYTE 5
BYTE 6	BYTE 7	BYTE 8
BYTE 9	BYTE 10	BYTE 11
BYTE 12	BYTE 13	BYTE 14
BYTE 15	BYTE 16	BYTE 17
BYTE 18	BYTE 19	BYTE 20
BYTE 21	BYTE 22	BYTE 23
BYTE 24	BYTE 25	BYTE 26

```
BYTE 27      BYTE 28      BYTE 29
BYTE 30      BYTE 31      BYTE 32
BYTE 33      BYTE 34      BYTE 35
BYTE 36      BYTE 37      BYTE 38
BYTE 39      BYTE 40      BYTE 41
BYTE 42      BYTE 43      BYTE 44
BYTE 45      BYTE 46      BYTE 47
BYTE 48      BYTE 49      BYTE 50
BYTE 51      BYTE 52      BYTE 53
BYTE 54      BYTE 55      BYTE 56
BYTE 57      BYTE 58      BYTE 59
BYTE 60      BYTE 61      BYTE 62
```

IL DISEGNO VIENE TRACCIATO SCRIVENDO:
1, 2 O 3 (A SECONDA DEL COLORE DESIDERATO)
NELLE CASELLINE DELLA MATRICE 12 X 21

I BYTE VENGONO RIEMPITI CON I BIT
CORRISPONDENTI A 1, 2, 3 (01, 10, 11)
E COMPLETANDO CON BIT 0

Figura 2.5 Schemi per la preparazione degli SPRITE MULTICOLORE

Le dimensioni dello sprite non variano dato che il COMMODORE 64 è stato costruito in modo che un punto multicolore ha la larghezza di due punti normali.

La gestione degli sprite multicolori risulta molto semplice; il registro 28 del VIC II (il 29-esimo) dà notizia, con il valore dei suoi 8 bit, di quali sprite devono essere visualizzati in modo multicolore, e di quali in modo normale (alta risoluzione). Nella Figura 2.5 è mostrata la corrispondenza tra i bit della matrice di 63 byte e i punti dello sprite multicolore.
I colori sono 4, di cui uno è quello dello sfondo, uno è quello contenuto nel registro del colore dello sprite, e gli altri due sono i colori contenuti in due registri di colore ausiliario, rispettivamente il numero 37 e il numero 38. Va notato che i due colori contenuti in questi registri sono i colori ausiliari per tutti gli 8 sprite visualizzabili.

Il programma SPRITE12, che ti presentiamo, mostra l'animazione di un atleta che corre. Abbiamo adottato la stessa tecnica del programma SPRITE8 per calcolare i valori da memorizzare per descrivere lo sprite, solo che ora per ANNERIRE ogni punto si usa un numero da 1 a 3, che indica il colore con il quale lo si vuole disegnare. Purtroppo in multicolore l'immagine disegnata non è così immediata come nel modo ad alta risoluzione (risulta più stretta); per ovviare a questo, la routine, che converte i dati grafici in numeri, stampa anche l'immagine dello sprite ingrandita e con i colori originali.

```
1 REM SPRITE12
5 REM ATLETA CHE CORRE
10 POKE56,36:CLR:POKE53281,5
20 DIMD(64),A$(3):VI=53248
30 A$(0)="  "
40 A$(1)=CHR$(18)+CHR$(144)+CHR$(32)+CHR$(146)
50 A$(2)=CHR$(18)+CHR$(158)+CHR$(32)+CHR$(146)
60 A$(3)=CHR$(18)+CHR$(129)+CHR$(32)+CHR$(146)
70 SP$="                                        "
100 FORNB=144TO150
110 GOSUB9000:GOSUB9100
120 NEXT
130 PRINTCHR$(147):FORI=1TO9:PRINT:NEXT
140 PRINTCHR$(18)CHR$(144)SP$;
150 PRINTCHR$(149);
160 FORI=1TO6:PRINTSP$;:NEXT
170 PRINTCHR$(18)CHR$(144)SP$;
190 POKEVI+1,140:REM POSIZIONE VERTICALE
200 POKEVI+23,1:REM ESPANDE IN ALTEZZA
210 POKEVI+28,1:REM SPRITE MULTICOLOR
219 REM COLORE MULTICOLOR #0 (NERO)
```

```
220 POKEVI+37,0
229 REM COLORE MULTICOLOR #1 (ARANCIO)
230 POKEVI+38,8
239 REM COLORE SPRITE #0 (GIALLO)
240 POKEVI+39,7
250 POKEVI+21,1:REM ACCENDE SPRITE #0
260 FORX=0TO6:FORI=0TO6
270 POKE2040,I+144:Z=(X*7+I)*7
280 POKEVI,ZAND255:POKEVI+16,Z/256
290 FORR=1TO150:NEXT:NEXT:NEXT
300 GOTO260
1000 DATA"            "
1010 DATA"            "
1020 DATA"     11     "
1030 DATA"     133    "
1040 DATA"     133    "
1050 DATA"     33     "
1060 DATA"     23     "
1070 DATA"    232     "
1080 DATA"    2322    "
1090 DATA"    2322    "
1100 DATA"    2322    "
1110 DATA"    2322    "
1120 DATA"    1311    "
1130 DATA"    1111    "
1140 DATA"     111    "
1150 DATA"     33     "
1160 DATA"     33     "
1170 DATA"     22     "
1180 DATA"     221    "
1190 DATA"     111    "
1200 DATA"            "
1210 DATA"            "
1220 DATA"            "
1230 DATA"     11     "
1240 DATA"     133    "
1250 DATA"     133    "
1260 DATA"     33     "
1270 DATA"   223322   "
1280 DATA"  3322223   "
1290 DATA" 33 22223   "
```

```
1300 DATA"  3322223    "
1310 DATA"   332223333"
1320 DATA"    2222     "
1330 DATA"    1111     "
1340 DATA"    11111    "
1350 DATA"   3331133   "
1360 DATA"12233   333 "
1370 DATA"1223     33 "
1380 DATA"1         22 "
1390 DATA"          22 "
1400 DATA"          111"
1410 DATA"             "
1420 DATA"             "
1430 DATA"             "
1440 DATA"     11      "
1450 DATA"     133     "
1460 DATA"     133     "
1470 DATA"     33      "
1480 DATA"    223      "
1490 DATA"    3222     "
1500 DATA"    3222     "
1510 DATA"    3222     "
1520 DATA"    333333   "
1530 DATA"    2222     "
1540 DATA"    1111     "
1550 DATA"    1111     "
1560 DATA"     111     "
1570 DATA"   12333     "
1580 DATA"   12333     "
1590 DATA"   1 22      "
1600 DATA"     22      "
1610 DATA"     111     "
1620 DATA"             "
1630 DATA"             "
1640 DATA"             "
1650 DATA"     11      "
1660 DATA"     133     "
1670 DATA"     133     "
1680 DATA"     33      "
1690 DATA"    222      "
1700 DATA"   32232     "
1710 DATA"  332232     "
```

```
1720 DATA"   32232    "
1730 DATA"    223333  "
1740 DATA"    2222    "
1750 DATA"    1111    "
1760 DATA"    1111    "
1770 DATA"   331111   "
1780 DATA"  333  333  "
1790 DATA" 233    233 "
1800 DATA"222    122  "
1810 DATA"12     12   "
1820 DATA" 11     1   "
1830 DATA"            "
1840 DATA"            "
1850 DATA"            "
1860 DATA"     11     "
1870 DATA"     133    "
1880 DATA"     133    "
1890 DATA"     33     "
1900 DATA"   222222   "
1910 DATA"  3322223   "
1920 DATA" 33 22223   "
1930 DATA"  3322223   "
1940 DATA"   322223333"
1950 DATA"    2222    "
1960 DATA"    1111    "
1970 DATA"    11111   "
1980 DATA"   331 133  "
1990 DATA"12233   333 "
2000 DATA"1223     33 "
2010 DATA"1        22 "
2020 DATA"         22 "
2030 DATA"         111"
2040 DATA"            "
2050 DATA"            "
2060 DATA"            "
2070 DATA"     11     "
2080 DATA"     133    "
2090 DATA"     133    "
2100 DATA"     33     "
2110 DATA"    222     "
2120 DATA"    2232    "
2130 DATA"    2232    "
```

```
2140 DATA"     2232     "
2150 DATA"     223333   "
2160 DATA"     2222     "
2170 DATA"     1111     "
2180 DATA"     1111     "
2190 DATA"      111     "
2200 DATA"    12333     "
2210 DATA"    12333     "
2220 DATA"    1 22      "
2230 DATA"      22      "
2240 DATA"      111     "
2250 DATA"              "
2260 DATA"              "
2270 DATA"              "
2280 DATA"      11      "
2290 DATA"      133     "
2300 DATA"      133     "
2310 DATA"      33      "
2320 DATA"     222      "
2330 DATA"    33222     "
2340 DATA"   332222     "
2350 DATA"    33222     "
2360 DATA"     332233   "
2370 DATA"     2222     "
2380 DATA"     1111     "
2390 DATA"     1111     "
2400 DATA"    331133    "
2410 DATA"   333  333   "
2420 DATA" 233     233 "
2430 DATA"222     122   "
2440 DATA"12      12    "
2450 DATA" 11      1    "
2460 DATA"              "
9000 PRINT"□":FORI=0TO20:READA$
9010 FORJ=0TO2:FORK=1TO4:B$=MID$(A$,J*4+K,1)
9020 R=VAL(B$):PRINTA$(R);
9030 BY=BY*4+R:NEXT
9040 D(I*3+J)=BY:BY=0:NEXT:PRINT:NEXT
9050 RETURN
9100 FORI=0TO62
9110 POKENB*64+I,D(I)
9120 NEXT:RETURN
```

COMMENTO A SPRITE12.
Linea 10: viene abbassato il top della memoria per riservare locazioni sopra la zona dedicata al BASIC.
Linee 30-60: la matrice A$ contiene 4 elementi; ciascuno di essi serve per stampare uno spazio di colore diverso. A$(0) per spazio nel colore dello sfondo. A$(1) per spazio nero in campo inverso. A$(2) per spazio giallo in campo inverso. A$(3) per spazio arancio in campo inverso.
Linea 70: SP$ contiene 40 spazi.
Linee 100-120: viene richiamata la routine che converte i dati e li memorizza.
Linee 130-170: disegna la pista per l'atleta.
Linee 190-250: prepara i registri per visualizzare lo sprite numero 0 in modo multicolore.
Linee 260-300: sposta l'atleta, cambia il puntatore ai dati (byte 2040) e genera i cicli di attesa necessari per produrre l'animazione.
Linee 1000-2260: linee DATA per preparare le immagini.
Linee 9000-9050: routine che legge 21 linee DATA, le stampa e le converte in 63 numeri che pone nella matrice D.
Linee 9100-9120: routine che pone i dati contenuti nella matrice D in memoria a partire dal byte di indirizzo NB*64. I dati degli sprite vengono memorizzati da 9216 a 9662.

2.5 RIASSUNTO DELLE CARATTERISTICHE DEGLI SPRITE

Il COMMODORE 64 può visualizzare fino a un massimo di 8 sprite.
Gli sprite possono essere di due tipi: alta risoluzione o multicolore.
Nelle Figure 2.1 e 2.5 è mostrata la corrispondenza tra punti e bit in memoria.
Il VIC II contiene diversi registri dedicati alla gestione degli sprite; essi iniziano alla locazione 53248. Gli 8 puntatori alle matrici di descrizione degli sprite si trovano negli 8 byte da 2040 a 2047.
Per visualizzare uno sprite devi:

1) disegnare lo sprite servendoti della griglia riportata nella Figura 2.1 o dei riferimenti indicati nella Figura 2.5;

2) convertire l'immagine in numeri, 63 numeri, come indicato nella Figura 2.1 o nella Figura 2.5;

3) memorizzare i 63 numeri che danno l'immagine dello sprite o nelle locazioni da 832 a 1023 (buffer della cassetta), che possono essere usate solo se si hanno fino a 3 sprite, o sopra il programma BASIC, come indicato nei programmi esempio, ricordando però che non si deve uscire dal banco di memoria utilizzato (inoltre la prima locazione di memorizzazione di uno sprite deve avere indirizzo multiplo di 64);

4) memorizzare negli appropriati registri:
posizione orizzontale X, registri pari da 0 a 14,
posizione verticale Y, registri dispari da 1 a 15,
bit più significativo della posizione X degli 8 sprite, registro 16,
abilitazione degli sprite, registro 21,
espansione verticale, registro 23,
espansione orizzontale, registro 29,
priorità sullo sfondo, registro 27,
selezione sprite multicolore, registro 28,
colore ausiliario numero 0, registro 37,
colore ausiliario numero 1, registro 38,
colore dello sprite, registri da 39 a 46;

5) utilizzare, se è il caso i due registri delle collisioni: il 30 per le collisioni tra sprite e il 31 per le collisioni tra sprite e sfondo.

Il movimento dello sprite si ottiene cambiandone la posizione dopo un opportuno intervallo di tempo.

Nei registri usati per tutti gli otto sprite, il bit di posizione 0 è dedicato allo sprite numero 0, il bit di posizione 7 allo sprite numero 7, e così via. Nel Capitolo 3 sono elencati i registri del VIC II.

I puntatori ai dati descrittivi degli sprite si trovano nei byte di indirizzo da 2040 a 2047, solo quando è attivo il banco numero 0 di memoria per il VIC II e la pagina video è posta all'indirizzo 1024 (0400H). In realtà il VIC II usa come puntatori ai dati degli sprite gli 8 byte che si trovano da VM + 1016 a VM + 1023 (VM = indirizzo primo byte memoria video. Se si cambia banco, cioè se si modificano i bit di posizione 0 e 1 del registro di indirizzo 56576, allora gli indirizzi dei puntatori sono modificati, sommando ai precedenti indirizzi il numero del banco moltiplicato per 16384. Analogamente gli indirizzi posti nei puntatori devono corrispondere alle locazioni dove sono stati memorizzati gli sprite nel banco di memoria attivo per il VIC II. Questi argomenti sono stati ampiamente trattati nel Volume 4.

Per esempio: il byte di indirizzo 56576 contiene 10 nei due bit meno significativi, cioè seleziona per il VIC II il banco numero 1, che corrisponde agli indirizzi da 16384 a 32767. I dati descrittivi degli sprite devono trovarsi nella zona di memoria da 16384 a 32767. I puntatori agli sprite si trovano nei byte di indirizzo 18424-18431 (16384+2040=18424). Se poniamo 14 nel byte di indirizzo 18424, puntatore allo sprite numero 0, questo significa che viene puntato l'indirizzo 16384+14*64=17226. Allo stesso modo variano gli indirizzi dove il VIC II legge le informazioni relative allo sfondo.

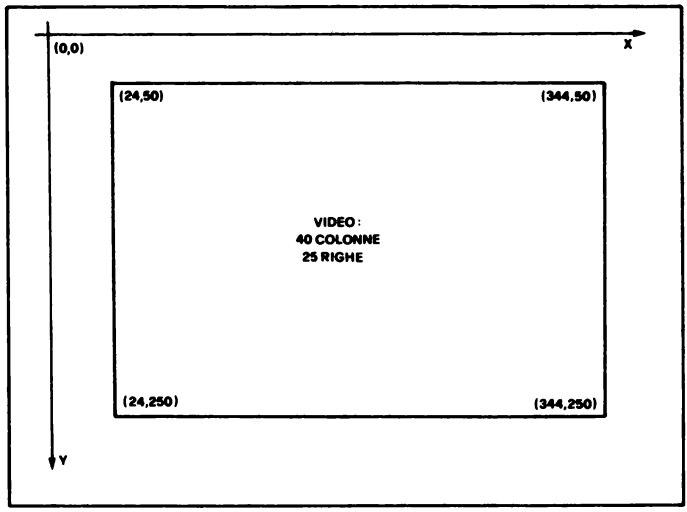


Figura 2.6 Coordinate per il posizionamento degli Sprite

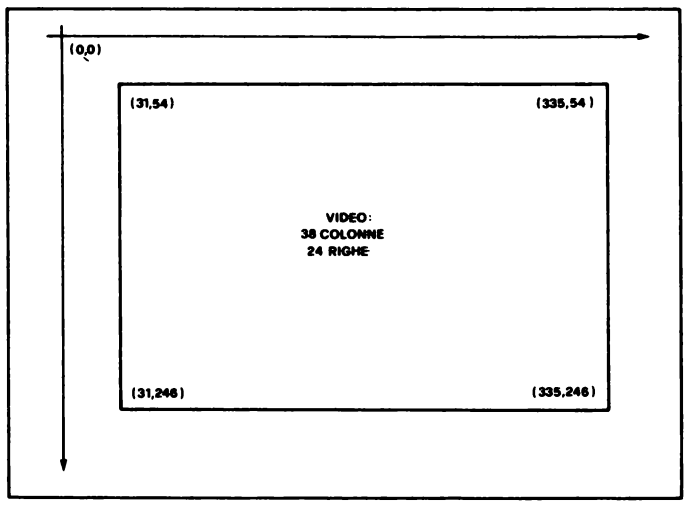


Figura 2.7 Coordinate per il posizionamento degli Sprite

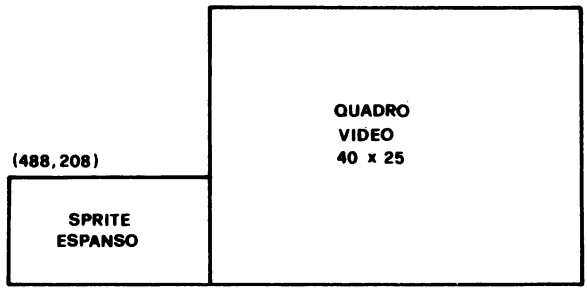


Figura 2.8 Posizione di uno SPRITE ESPANSO

CAPITOLO 3

IL SUONO

3.1 PERCHÈ IL SUONO

Il tuo COMMODORE 64 è in grado di produrre suoni; non solo note senza alcun carattere, ma può riprodurre strumenti reali e generare suoni non producibili da alcuno strumento musicale. Così come rende possibile la simulazione dei più diversi rumori. Prima di iniziare ad esplorare tutte le capacità sonore del tuo calcolatore, vogliamo soffermarci un attimo per cercare di rispondere a una domanda: perchè dotare un calcolatore di capacità sonore?
Forse, la prima cosa che viene in mente sono i videogiochi: se a delle buone immagini si associano degli effetti sonori realistici allora il videogioco diventa decisamente più interessante e divertente. Il COMMODORE 64 è in grado di offrire una grafica eccezionale con degli effetti sonori ottimi.
Ma non è questo tutto ciò per cui si può utilizzare il suono.
In programmi comuni esso può servire per il cosiddetto AUDIO FEEDBACK, cioè per segnalare all'utente che il calcolatore ha svolto una determinata azione. Il suono può essere usato per indicare che i dati ricevuti sono errati, oppure che il programma è in attesa di comandi, o ancora, il termine di una lunga elaborazione, per segnalare che i risultati sono pronti, o semplicemente come SUONERIA per un orologio computerizzato. La possibilità di produrre suoni a programma dà all'utente una maggiore libertà di azione; mentre il calcolatore lavora egli può fare altre cose, a un certo punto riceve un avviso e provvede a intervenire.
Se poi ti interessa la musica, con il tuo COMMODORE 64 hai a disposizione 3 generatori indipendenti per creare melodie di tutti i generi; se il diffusore del televisore o del monitor non ti sembra sufficiente per i suoni che vuoi generare, puoi collegare il calcolatore al tuo stereo.
Inoltre suoni ed effetti speciali possono essere ottenuti con delle POKE! Occupiamoci quindi di questo argomento.

3.1.1 Come si genera un suono

L'utilità di dotare un microcalcolatore della capacità di emettere suoni è ormai riconosciuta, ma non tutti lo fanno.
Esistono almeno due modi per permettere a un calcolatore di suonare:

1) Metodo software: il calcolatore è dotato di un altoparlante interno, collegato a un registro o una porta di I/O (cioè un indirizzo al quale si accede per comunicare con dispositivi esterni). Per generare un suono si devono scrivere ripetutamente nelle locazioni adatte valori che creino una sequenza di impulsi, che inviati all'altoparlante ne fanno vibrare la membrana e quindi produrre un suono. Vedi la Figura 3.1.

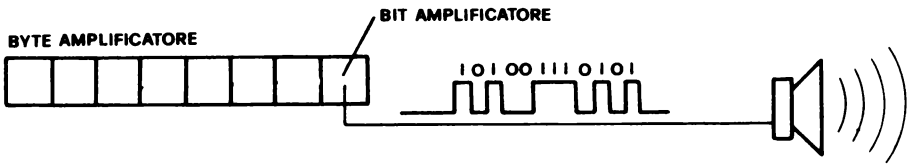


Figura 3.1 Generazione di un suono

2) Metodo hardware: il calcolatore è equipaggiato con una serie di circuiti o con un integrato programmabile in grado di ricevere i dati relativi al suono da produrre e quindi generare un segnale da miscelare al segnale video, e/o un segnale da inviare ad un altoparlante, e/o un segnale prelevabile con apposita presa ed inviabile al proprio HI-FI.
Altri metodi si possono ottenere da una fusione dei due e sono usati in generale nei SINTETIZZATORI e nei CAMPIONATORI.
Per il COMMODORE 64 è stato scelto il secondo metodo. Innanzi tutto esso permette una maggior comodità e velocità nella generazione dei suoni, poichè, una volta selezionati i parametri necessari, è l'integrato che si preoccupa di tutto e il programma può proseguire.
Esso inoltre è generalmente più versatile e più potente. Lo svantaggio è naturalmente nell'aumento del costo del calcolatore, che però la COMMODORE è in grado di evitare, dato che l'integrato sonoro, di codice 6581 e nome SID, è CUSTOM, cioè prodotto dalla ditta per i suoi scopi.
L'accesso alla programmazione del SID è ottenuta facendo in modo che la scrittura in determinati byte ponga i valori scritti nei registri interni del SID, utilizzando la tecnica di MAPPATURA IN MEMORIA del dispositivo, che abbiamo già avuto modo di vedere per il VIC II.

3.1.2 Onde sonore, frequenza, volume e timbro

Immagina di lanciare un sasso in una pozza di acqua; dal punto dove il sasso entra in contatto con il liquido si irradiano delle onde circolari. Viste di profilo queste onde appaiono più o meno come indicato nella Figura 3.2.

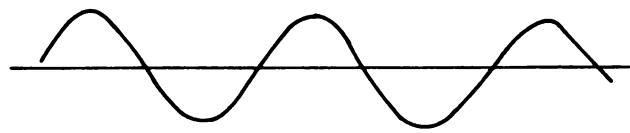


Figura 3.2 Profilo onda sonora

Il suono, come noi lo percepiamo è prodotto da ONDE SONORE che si irradiano da una SORGENTE SONORA in tutte le direzioni.
Mentre le onde dell'acqua mettono il liquido in movimento, le onde sonore sono di COMPRESSIONE e RAREFAZIONE dell'aria, ma si comportano più o meno nello stesso modo delle altre onde; essendo nello spazio la irradiazione risulta su una superficie sferica rispetto alla sorgente.

Se prendi un elastico, lo tendi e lo pizzichi, puoi variare il suono prodotto in 3 modi:

- pizzicando più forte,
- modificando la tensione,
- cambiando l'elastico.

Supponiamo che l'onda sonora prodotta nell'aria dalla vibrazione dell'elastico sia rappresentabile come indicato nella figura 3.3, dove:

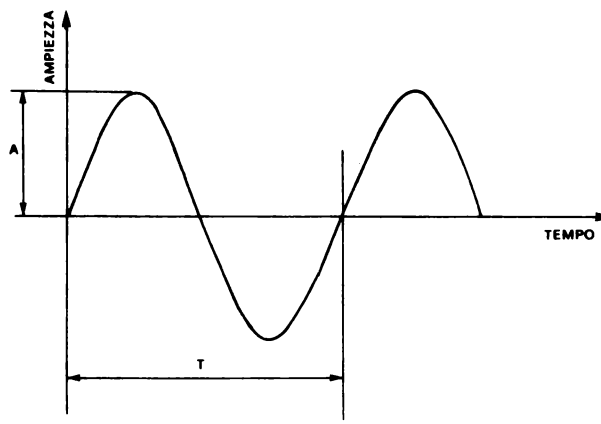


Figura 3.3 Rappresentazione onda sonora

• A, il valore massimo dell'onda, è chiamato AMPIEZZA DELL'ONDA. Se pizzichi l'elastico più forte o più piano, l'ampiezza dell'onda aumenta o diminuisce in conseguenza (essa determina il volume del suono).

• T è il tempo che intercorre tra due PICCHI delle onde, meglio, il tempo che passa perchè avvenga una vibrazione completa. Per avere il numero di vibrazioni in un secondo basta calcolare 1/T. Questo valore è detto FREQUENZA DELL'ONDA. Se T è espresso in secondi, F=1/T è espresso in Hz (hertz). Quando tendi o allenti l'elastico è questa la caratteristica del suono che cambia.

• L'ultima cosa da capire è perchè cambiando elastico e lasciando invariata la tensione e il pizzico, il suono non è uguale al precedente. Il motivo è che i due elastici non sono perfettamente uguali, così come differiscono le corde di una chitarra da quelle di un pianoforte. La caratteristica che ci permette di distinguere due sorgenti sonore, quando i suoni da esse prodotti hanno la stessa frequenza e lo stesso volume, si chiama TIMBRO. Vedremo più avanti come definirlo con IL COMMODORE 64.

3.1.3 Suoni con il Commodore 64

Il centro del sistema sonoro del COMMODORE 64 è un integrato programmabile detto SID (Sound Interface Device, cioè dispositivo di interfaccia per il suono). Come per il VIC II esiste una zona di memoria riservata al SID, ove si possono effettuare le POKE per ottenere i suoni desiderati. Tale zona inizia alla locazione 54272 (D400H) ed è composta da 29 registri, cioè i byte in cui effettuare le POKE. Il SID mette a disposizione 3 diversi canali o VOCI per produrre suoni, che sono totalmente indipendenti tra loro. Per ognuna delle voci sono riservati 7 registri e gli ultimi 8 sono comuni alle 3 voci.
In particolare il registro numero 24 (il 25-esimo), cioè quello all'indirizzo 54296, controlla il volume di tutte le tre voci. Il valore assoluto del volume può variare tra 0 (bassissimo, non elimina completamente il suono) e 15 (il massimo, molto spesso si pone il volume a 15 e poi lo si regola più finemente con il volume del proprio televisore) e la sua variazione si ottiene modificando i bit 0-3 del registro indicato, come puoi vedere nella Figura 3.4, e lasciando inalterati i restanti bit.

BIT	7	6	5	4	3	2	1	0
REG. N.24 IND. 54296	X	X	X	X	V3	V2	V1	V0

Figura 3.4 Registro controllo volume

Se V è il volume e VP il valore precedentemente posto in questo byte:

POKE 54296,VP AND 240 + V

seleziona il volume desiderato.

Torniamo adesso un attimo alla caratteristica del suono chiamata frequenza. L'orecchio umano riesce a percepire suoni la cui frequenza sia compresa tra 20 e 20000 Hz. Il SID è in grado di generare suoni di frequenza compresa tra 0 Hz e 4000 Hz (cioè può generare anche suoni sotto la soglia uditiva umana).
Per indicare al SID la frequenza del suono che si vuole ottenere si deve trasformarla in un numero a 16 bit, compreso tra 0 e 65535. Sono i primi due registri di ogni voce che contengono il byte basso e il byte alto di questo numero a 16 bit. Poichè il valore massimo della frequenza è 4000 Hz e i numeri rappresentabili con 16 bit sono da 0 a 65535, il numero da scrivere in memoria è:

N=Frequenza/(4000/65535)=Frequenza/0.06097
(ovvero N=Frequenza*16.3835)

il che vuol dire che la variazione di N di una unità equivale ad una variazione della frequenza di soli 0.06 Hz.
Ora dobbiamo trovare gli 8 bit più significativi (byte alto) e i meno significativi (byte basso) del numero N:

NHI=INT(N/256) e NLO=N-NHI*256

questi due valori vanno posti negli indirizzi indicati nella Tabella 3.1.

BYTE	VOCE1	VOCE2	VOCE3
NHI	54273	54280	54287
NLO	54272	54279	54286

Tabella 3.1 Indirizzi byte registri frequenza

Nel Paragrafo 3.6 riportiamo una tabella dei valori utilizzabili per generare quasi 8 ottave di note.
Per calcolarli ci si è basati sul presupposto che la frequenza di ogni nota è circa pari alla radice 12-esima di 2 per la frequenza della nota precedente. Questa è la differenza di frequenza che esiste tra due semitoni, poichè nella scala temperata tra la nota DO1 e la nota DO2 ci sono 12 semitoni e la frequenza di DO2 è doppia di quella di DO1:

```
Freq(DO2)=(2↑12)↑(1/12)*Freq(DO1)=2*Freq(DO1)
```

3.1.4 Forme d'onda e ADSR, il timbro

Il timbro che caratterizza un suono è determinato dalla FORMA DELL'ONDA che si propaga nell'aria, pertanto volume e frequenza non sono sufficienti al SID per produrre un suono; è necessario indicare quale tipo di forma d'onda utilizzare. Il SID mette a disposizione 4 forme d'onda, come indicato nella Figura 3.5.

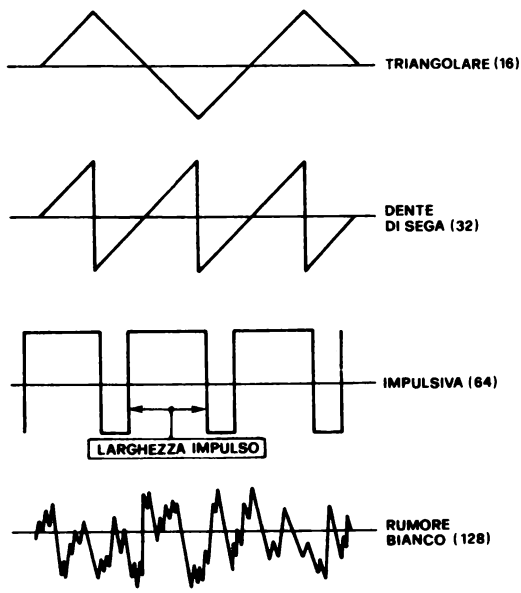


Figura 3.5 Forme d'onda del SID

Per selezionarle devi porre a 1 un determinato bit nel quinto registro della voce che vuoi usare, secondo lo schema che viene riportato nella Figura 3.6.

	VOCE1	VOCE2	VOCE3
FORMA D'ONDA	54276	54283	54290

UTILIZZO BIT							
7	6	5	4	3	2	1	0
●	●	●	●				

```
 |     |     |     |
 |     |     |     |
 |     |     |     ---->ONDA TRIANGOLARE
 |     |     ------->ONDA DENTE DI SEGA
 |     ------------->ONDA IMPULSIVA
 ------------------->RUMORE BIANCO
BIT=1 PER ON
BIT=0 PER OFF
```

Figura 3.6 Selezione forma d'onda

Se poi selezioni l'onda impulsiva, devi indicare al SID che percentuale dell'onda deve essere positiva; vedi, come riferimento, la Figura 3.7.

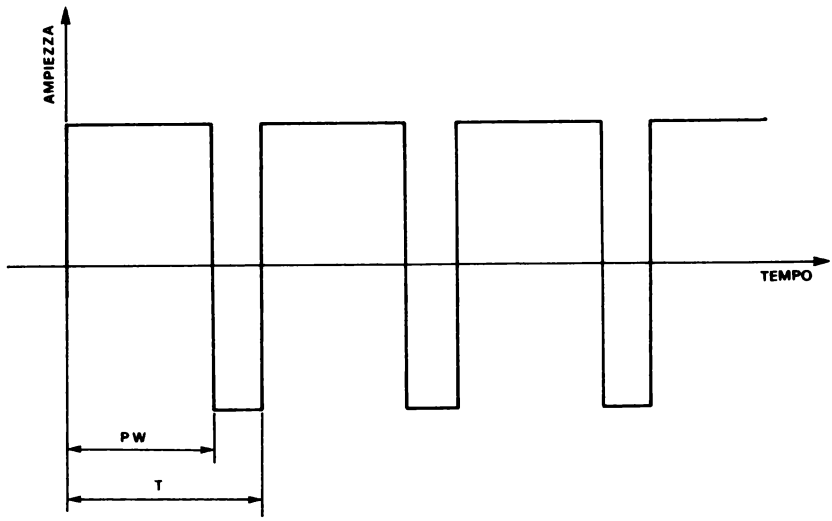


Figura 3.7 Parte positiva impulso

Il rapporto PW=P/T*4095, sotto forma di numero a 12 bit, va trascritto nel terzo e quarto registro per la voce che usi, gli 8 bit meno significativi nel terzo e i 4 più significativi nel quarto. Il valore che si ottiene è:

PW=PWHI * 256 + PWLO

ed in uscita la percentuale positiva dell'impulso è:

(PW/40.95)%

Se PW=0, solo livello basso, o PW=4095 solo livello alto, l'uscita dell'altoparlante sarà nulla. Se PW=2047, cioè PWHI=8 e PWLO=0, l'onda è quadra.
Se R è la percentuale positiva dell'onda impulsiva, allora PW=R*40.95, e le POKE da fare, ad esempio, nel caso della prima voce sono:

POKE 54274, PWLO
POKE 54275, PWHI

Gli indirizzi dei registri da usare in questo caso sono riportati nella Tabella 3.2.

BYTE	VOCE1	VOCE2	VOCE3
PWHI	54275	54282	54289
PWLO	54274	54281	54288

Tabella 3.2 Indirizzi byte dei registri per la parte positiva dell'onda

Vediamo perchè cambiando la forma d'onda cambia il suono.

Quando senti una nota essa è generata da un'onda sinusoidale, la cui frequenza è la frequenza del suono e ne definisce in pratica la tonalità (frequenza fondamentale). Contemporaneamente sono generate altre onde, dette ARMONICHE, la cui caratteristica è avere frequenza multipla per un intero della frequenza fondamentale. Un'onda sonora perciò è la somma della frequenza fondamentale più tutte le armoniche necessarie per produrla.
A ogni armonica è associato l'intero che dà il rapporto con la frequenza fondamentale, cioè l'armonica numero 1. La seconda armonica ha frequenza doppia, la settima armonica ha frequenza sette volte la fondamentale e così via.
Il timbro, cioè la FORMA DELL'ONDA, dipende dalle armoniche che costituiscono il suono e dall'ampiezza di ognuna di esse, quindi tali valori saranno diversi per diverse corde vibranti.
Per le FORME D'ONDA a disposizione valgono queste regole:

- ONDA TRIANGOLARE: solo le armoniche dispari; l'ampiezza di ogni armonica è proporzionale al reciproco del quadrato del numero dell'armonica, cioè, ad esempio, l'armonica numero 5 è ampia 1/25 dell'armonica fondamentale.
- ONDA DENTE DI SEGA: tutte le armoniche; l'ampiezza di ogni armonica è proporzionale al reciproco del numero dell'armonica.
- ONDA QUADRA: solo le armoniche dispari; l'ampiezza di ogni armonica è proporzionale al reciproco del numero dell'armonica (per rapporti R diversi da 50%, cioè per onde rettangolari generiche, la composizione dell'onda impulsiva varia notevolmente).
- RUMORE BIANCO: tutte le armoniche; ampiezza costante per tutte le armoniche.

Per i suoni reali la struttura armonica (cioè le armoniche presenti e la loro ampiezza) è molto più complessa. Per poter ottenere dei suoni più vicini ai reali si può ricorrere alla tecnica di filtrare le frequenze superiori o inferiori ad una certa frequenza data; lo vedremo più avanti.

Un'altra caratteristica che distingue un suono da un altro è il modo di variare di ampiezza nel tempo. Questa caratteristica è detta INVILUPPO del suono (ENVELOPE). Se premi un tasto del pianoforte il suono raggiunge istantaneamente la massima ampiezza ed altrettanto velocemente si smorza, se percuoti un triangolo il suono si smorza più lentamente.
Il SID dispone di un GENERATORE DI INVILUPPO (ENVELOPE GENERATOR) indipendente per ogni voce. Esso permette di descrivere l'andamento del volume del suono dividendolo in 4 fasi:

- A: ATTACCO (ATTACK), è il tempo che il suono impiega per raggiungere il massimo del volume. Con il SID puoi definirlo da 2 msec a 8 sec.
- D: DECADIMENTO (DECAY), è il tempo che il suono impiega, dopo aver raggiunto il picco per calare di intensità fino a zero, oppure fino a un dato livello chiamato di SOSTENIMENTO; da 6 msec a 24 sec.
- S: SOSTENIMENTO (SUSTAIN), è il livello di volume al quale il suono si assesta dopo la fase di decadimento. Puoi mantenere i suoni a questo livello finchè occorre e poi, con una POKE, iniziare la fase di RILASCIO.
- R: RILASCIO (RELEASE), è il tempo che il suono impiega per raggiungere il volume zero. Anche per il rilascio i valori variano da 6 msec a 24 sec.

Molto spesso si fa cominciare una nuova nota prima che la nota precedente sia completamente smorzata. Ciò consente anche, usando adeguatamente le 3 voci, di creare dei veri e propri inseguimenti tra le note.
Nella Figura 3.8 riportiamo il grafico delle 4 fasi e l'inviluppo.

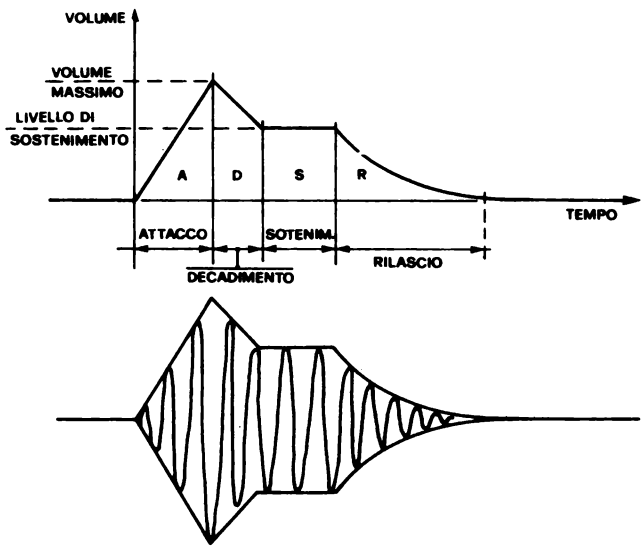


Figura 3.8 Grafico delle 4 fasi e inviluppo

Prima di vedere come indicare al SID il tipo di inviluppo da usare, vogliamo farti notare che puoi ottenere lo stesso effetto utilizzando soltanto il controllo del volume, ma che con questo metodo NON PUOI VARIARE L'ANDAMENTO DEL VOLUME DI OGNI SUONO IN MODO INDIPENDENTE.

Per ognuna delle voci, il sesto e il settimo registro controllano l'ADSR (Attacco, Decadimento, Sostenimento, Rilascio).
Nel sesto registro vanno memorizzati dei valori, compresi tra 0 e 15, per selezionare i tempi di attacco e decadimento secondo la Tabella 3.3.

VALORE	ATTACK TEMPO	DEC/REL TEMPO	SUSTAIN %VOLUME
0	2 MS	6 MS	0.0
1	8 MS	24 MS	6.7
2	16 MS	48 MS	13.3
3	24 MS	72 MS	20.0
4	38 MS	114 MS	26.7
5	56 MS	168 MS	33.3
6	68 MS	204 MS	40.0
7	80 MS	240 MS	46.7
8	100 MS	300 MS	53.3
9	250 MS	750 MS	60.0
10	500 MS	1.5 S	66.7
11	800 MS	2.4 S	73.3
12	1 S	3.0 S	80.0
13	3 S	9.0 S	86.7
14	5 S	15.0 S	93.3
15	8 S	24.0 S	100.0

NOTA: MS PER MILLISECONDI
S PER SECONDI

Tabella 3.3 Valori da porre nel sesto e nel settimo registro

Nel settimo registro vanno memorizzati i valori relativi al sostenimento e al tempo di rilascio; anche questi valori sono riportati nella Tabella 3.3.
Nella Figura 3.9 viene mostrato l'utilizzo dei bit dei due registri.

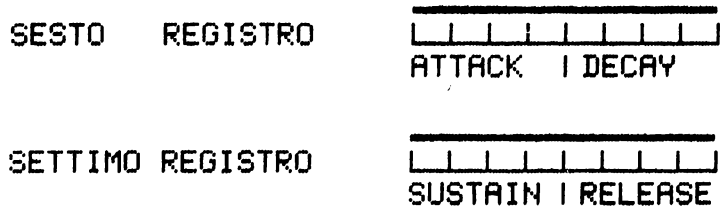


Figura 3.9 Utilizzo sesto e settimo registro

Gli indirizzi dei registri sono mostrati nella Tabella 3.4.

BYTE	VOCE1	VOCE2	VOCE3
A/D	54277	54284	54291
S/R	54278	54285	54292

Tabella 3.4 Indirizzi registri ADSR

La procedura tipica per generare una nota è:

- selezionare ATTACCO, DECADIMENTO, SOSTENIMENTO e RILASCIO per la nota;
- selezionare le frequenze desiderate ed eventualmente la percentuale dell'onda impulsiva;
- selezionare la forma d'onda e mettere a 1 il primo bit dello stesso registro; questo bit, detto GATE BIT, indica che la voce è selezionata per suonare e quando viene riportato a 0 fa iniziare la fase di rilascio. Nella Figura 3.10 sono anche indicati i valori da porre nei registri indicati per le 4 forme d'onda.

	VOCE1	VOCE2	VOCE3
FORMA D'ONDA	54276	54283	54290

UTILIZZO BIT							
7	6	5	4	3	2	1	0
0	0	0	0	-	-	-	●

TIPO DI ONDA

->GATE BIT

VALORI PER POKE

FORMA D'ONDA	INIZIO ATTACCO	INIZIO RILASCIO
TRIANGOLARE	17	16
DENTE DI SEGA	33	32
IMPULSIVA	65	64
RUMORE BIANCO	129	128

Figura 3.10 Valori per selezionare la forma d'onda

Con un uso appropriato di questo controllo puoi creare effetti sonori più complessi poichè ogni volta che il bit passa da 1 a 0 inizia la fase di rilascio e se passa da 0 a 1 inizia la fase di attacco. I grafici della Figura 3.11 possono servire di chiarimento.

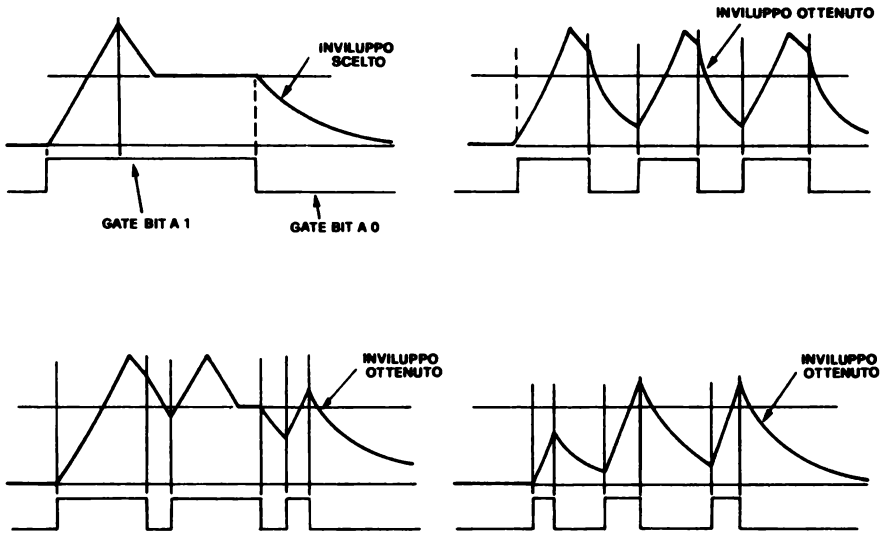


Figura 3.11 Inviluppi

Per sfruttare appieno queste possibilità è però necessario programmare in linguaggio macchina.
Quando vuoi terminare la nota, poni a 0 il GATE BIT.

3.2 PROGRAMMI SONORI

Ora hai tutti gli elementi per comprendere come funziona un programma sonoro. Utilizzando una sola voce è sufficiente, una volta selezionati la forma d'onda e l'inviluppo, fornire la durata e la frequenza di ogni nota, per far suonare il tuo COMMODORE 64.
Ogni coppia di valori XXX,YYYY, che di norma memorizzerai in linee DATA, avrà questo significato:

XXX rappresenta la durata del suono; esso può essere interpretato come parametro finale di un ciclo di conteggio, oppure come numero di 16-esimi o 64-esimi per i quali deve essere tenuta attiva la nota;
YYYY frequenza del suono; risulta più comodo fornirla come 2 numeri che sono i valori da memorizzare nei 2 byte che specificano la frequenza del suono per le varie voci.

Per indicare una pausa si può, ad esempio, usare la frequenza zero o porre la durata e la frequenza come numeri negativi.
Un altro sistema per memorizzare questi dati è di comporre con essi un numero seguendo la regola riportata nella Figura 3.12, esprimendo i parametri in binario e combinandoli insieme.

DURATA	D
3/2	0
1	1
3/4	2
1/2	3
3/8	4
1/4	5
3/16	6
1/8	7
3/32	8
1/16	9
3/64	10
1/32	11
3/128	12
1/64	13

OTTAVA	O
1^	0
2^	1
3^	2
4^	3
5^	4
6^	5
7^	6
8^	7

NOTA	N
PAUSA	0
DO(SI♯)	1
DO♯(REb)	2
RE	3
RE♯(MIb)	4
MI	5
FA	6
FA♯(SOLb)	7
SOL	8
SOL♯(LAb)	9
LA	10
LA♯	11
SI	12

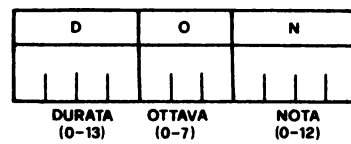


Figura 3.12 Regole per il calcolo dei parametri del suono

In questo caso il numero non supererà mai le 4 cifre decimali (sarà compreso tra 0 e 8191).
Se non devi scendere sotto 1/16 per la durata delle note puoi usare lo stesso sistema:

Valore=((((D*8)+O)*16)+N)
Dove:
D=Durata in 1/16 (di una misura); se negativo indica pausa,
O=Ottava (compresa tra 0 e 7),
N=Nota, tra 0 e 11 o 1 e 12, come preferisci.

Per ricavare i parametri di ogni nota si procede all'inverso; se hai prelevato da linee DATA un VALORE:

Durata=INT(Valore/128)
Ottava=INT((Valore-Durata*128)/16)
Nota= Valore - Durata*128 - Ottava*16

Ti proponiamo, come esempio, il programma SUONO4.

```
1 REM SUONO4
10 REM FARE MUSICA
100 S=54272:FORI=STOS+24:POKEI,0:NEXT
110 DIMF(11)
120 POKES+3,8:REMPOKES+22,128:POKES+23,244
130 F(11)=67280:FORI=10TO0STEP-1
140 F(I)=INT(F(I+1)/2↑(1/12)+.5)
150 NEXT
160 O=5
170 POKES+5,41:POKES+6,0
180 POKES+24,31
190 READN$,T:IFN$="*"THEN430
200 PRINTN$;T;
210 IFN$="O"THENO=T:GOTO190
220 IFN$="O+"THENO=O+1AND7:GOTO190
230 IFN$="O-"THENO=O-1AND7:GOTO190
240 IFN$="R"THENRESTORE:GOTO190
250 IFN$="DO"THENN=0
260 IFN$="DO#"THENN=1
270 IFN$="RE"THENN=2
280 IFN$="RE#"THENN=3
290 IFN$="MI"THENN=4
300 IFN$="FA"THENN=5
310 IFN$="FA#"THENN=6
320 IFN$="SOL"THENN=7
```

```
330 IFN$="SOL#"THENN=8
340 IFN$="LA"THENN=9
350 IFN$="SI"THENN=11
360 F=F(N)/2↑(7-O)
365 IFF>65535THENF=0:REM SI-7
370 POKES,F-INT(F/256)*256
380 POKES+1,F/256
390 POKES+4,64:POKES+4,65
400 FORR=1TO2↑T*5:NEXT
420 GOTO190
430 FORL=STOS+24:POKEL,0:NEXT
440 DATAO-,0,DO,3,RE,2,MI,3,SOL,3,SOL,3,LA
445 DATA2,SOL,3,MI,4,DO,2,RE,3,MI,3,MI,3,RE,2
450 DATADO,2,RE,6,DO,3,RE,2,MI,3,SOL,3,SOL
455 DATA3,LA,2,SOL,3,MI,4,DO,2,RE,3,MI,3,MI,3
460 DATARE,2,RE,2,DO,6,FA,5,FA,5,LA,3,LA
465 DATA5,LA,2,SOL,3,SOL,3,MI,3,DO,3,RE,6
470 DATADO,3,RE,2,MI,3,SOL,3,SOL,3,LA,2,SOL
475 DATA3,MI,4,DO,2,RE,3,MI,3,MI,3,RE,2,RE,2
480 DATADO,6,R,4
```

COMMENTO A SUONO4.

Linea 100: inizializza la variabile S con l'indirizzo del SID.

Linea 110: il vettore F viene usato per contenere i valori da porre nel SID per generare i semitoni più alti.

Linea 120: pone la larghezza dell'impulso dell'onda quadra a metà ciclo e il valore del filtro a 3520 Hz.

Linea 130: F(11) contiene il numero da porre nel SID per generare il SI dell'ottava più alta (ottava 7); tale valore supera 65535, perchè nella versione europea la nota più alta che il SID può generare è il LA #. La linea 365 porrà a 0 la frequenza, nel caso in cui si voglia suonare un SI nell'ottava 7.

Linee 140-150: il valore della frequenza di un semitono si ricava da quello del semitono superiore, diviso per la radice 12-esima di 2.

Linea 160: pone come ottava di default l'ottava 5.

Linea 170: pone ATTACK=2, DECAY=9, SUSTAIN=0 e RELEASE=0.

Linea 180: pone a 15 il volume e aziona il filtro passa basso.

Linea 190: N$ e T rappresentano i valori della nota e la sua durata; N$, oltre ai nomi delle 7 note può assumere i significati:

O+, passaggio all'ottava superiore,

O-, passaggio all'ottava inferiore,

O, passaggio all'ottava specificata,
R, ripetizione dall'inizio,
*, termine dell'esecuzione.

Linea 200: evidenzia sul video il valore della nota corrente.
Linee 210-350: assegna a N il valore del semitono che corrisponde alla nota N$, o esegue il comando che N$ rappresenta.
Linea 360: calcola la frequenza in base alla nota e all'ottava; ogni nota ha infatti la metà della frequenza della stessa nota nell'ottava superiore.
Linea 365: se la frequenza della nota è troppo alta per il SID, allora viene annullata. Tale caso si verifica solo con il SI dell'ottava 7.
Linee 370-380: pone nei registri della frequenza i valori calcolati.
Linea 390: suona la nota.
Linea 400: attende per un tempo variabile, che dipende da T. T rappresenta la durata della nota, secondo questi criteri:

T=0, un 32-esimo
T=1, un 16-esimo
T=2, un ottavo
T=3, un quarto
T=4, un mezzo
T=5, un intero.

Linea 420: va a suonare la prossima nota.
Linea 430: termina l'esecuzione azzerando i registri del SID.
Linee 430-470: linee DATA che contengono un semplice motivo. Il formato dei dati è il seguente: nome della nota, tempo. Le note possono avere il diesis; se si introduce al posto della nota un comando, che non ha bisogno del parametro tempo, bisogna ugualmente porre un numero come tempo, per consentire la lettura dei dati in coppia.

Utilizzando più voci insieme è possibile ottenere suoni più pieni, accordi e l'impressione di orchestra poichè ogni voce può simulare uno strumento diverso. L'unico problema vero è quello della sincronizzazione delle 3 voci.
Anche in questo caso le soluzioni sono molte e diverse. La più pratica sembra però quella descritta anche nella Programmer's Reference Guide, cioè, per ognuna delle 3 voci, preparare una descrizione di ciò che avviene ogni 16-esimo di battuta (o ottavo o 64-esimo, dipende dall'uso che ne vuoi fare), quindi, stabilita la durata del 16-esimo, si legge la sequenza dei dati e si trascrivono nei registri delle 3 voci.

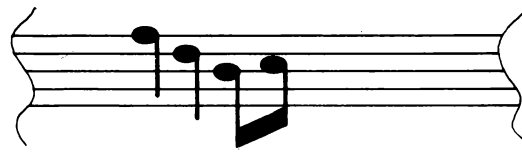


Figura 3.13 Sequenza di note

Ad esempio, se il pezzo da suonare con una voce è quello riportato nella Figura .13, le note sono: FA2, RE2, SI2, DO2 (il numero indica l'ottava), le durate 1/4, 1/4, 1/8, 1/8 (cioè 4/16, 4/16, 2/16, 2/16) e si ottiene uno schema in 16-esimi, come indicato nella Figura 3.14.

SEDICESIMI	NOTA	OTTAVA	ULT. SED.
1	FA	4	
2	FA	4	
3	FA	4	
4	FA	4	•
5	RE	4	
6	RE	4	
7	RE	4	
8	RE	4	•
9	SI	3	
10	SI	3	•
11	DO	4	
12	DO	4	•

Figura 3.14 Schema in sedicesimi di una sequenza di note

Esso può essere codificato con 12 triplette di numeri che indichino i valori da porre nei byte che definiscono la frequenza (2 byte) e il valore da porre nel byte che determina la forma d'onda. Dopo l'ultimo 16-esimo di ogni nota (nello schema sono indicati con un puntino) la nota verrà fatta terminare scrivendo nel registro che regola la forma d'onda un valore che ponga a 0 il GATE BIT della voce specificata (con un AND 254 del valore da scrivere nel registro apposito).

La fase più noiosa di tutto ciò sta nella codifica delle note che, già semplificata con i due metodi visti all'inizio del paragrafo, può essere ulteriormente agevolata costruendo un programma che accetti i dati per le varie voci dall'utente, ne consenta modifiche, cancellazioni, memorizzazione e ricaricamento, e li trasformi quindi:

- o nei dati organizzati in 3 vettori (byte alto frequenza, byte basso frequenza, forma d'onda), che è la più comoda se si utilizzano le 3 voci insieme;
- in istruzioni più semplici ed eseguite più rapidamente da una parte del programma che si occupa solo di suonare.

Entrambi i sistemi vengono notevolmente migliorati dall'uso del linguaggio macchina.

3.3 L'USO DEI FILTRI

Abbiamo visto che nella definizione di un suono entrano in gioco molti parametri. In particolare scegliendo una forma d'onda piuttosto che un'altra (e con l'onda rettangolare ne puoi ottenere molte tutte diverse), cambia la struttura armonica dell'onda sonora generata. L'uso della tecnica chiamata FILTERING (in italiano diventa filtraggio) permette di eliminare SELETTIVAMENTE le frequenze intorno, al di sopra o al di sotto di una frequenza base, detta frequenza di taglio (o rottura). Vediamo di spiegarci meglio: un FILTRO è un circuito che, ricevendo in ingresso un segnale è in grado di trasmetterlo (riprodurlo) modificato, eliminando un certo gruppo di frequenze che costituiscono il segnale oppure togliendo la parte di segnale di frequenza superiore o inferiore ad un certo limite, o altre cose del genere. Le caratteristiche che distinguono i diversi tipi di filtro sono:

- la frequenza di taglio,
- il tipo di filtro,
- il grado di smorzamento delle altre frequenze.

TIPO DI FILTRO indica quali frequenze, rispetto a quella di taglio devono essere eliminate, questa classificazione dà i filtri passa basso, passa alto, passa banda, taglia banda (o a tacca, notch in inglese) come indicato nella Figura 3.15.

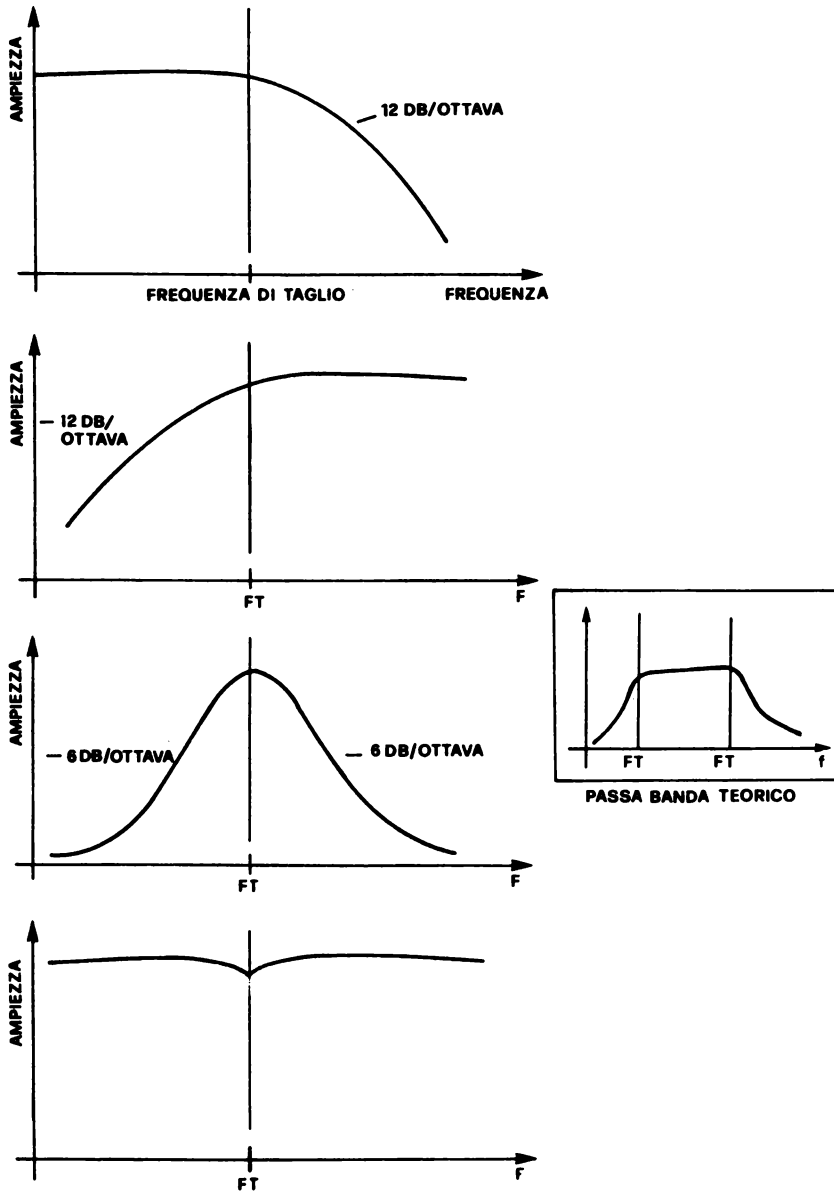


Figura 3.15 Tipi di filtri

Il COMMODORE 64 è equipaggiato con i filtri: passa basso, passa alto e passa banda, che, essendo adittivi, possono essere combinati per ottenere il filtro a tacca. Nella Figura 3.15 sono indicati i grafici dei 4 tipi di filtri.
Nel FILTRO PASSA BASSO vengono attenuate le frequenze al di sopra di quella di taglio. Nel FILTRO PASSA ALTO vengono attenuate le frequenze al di sotto di quella di taglio. Nel FILTRO PASSA BANDA vengono attenuate sia le frequenze al di sopra che quelle al di sotto della frequenza di taglio. Nel FILTRO A TACCA vengono attenuate la frequenza di taglio e quelle nel suo intorno; esso si ottiene attivando contemporaneamente i filtri passa basso e passa alto.

La PENDENZA delle curve dei grafici indica il grado di attenuazione delle frequenze da escludere. Per il SID l'attenuazione è di 12 db/ottava per i filtri passa alto e passa basso e di 6 db/ottava per il filtro passa banda. Osserva in proposito le note della Figura 3.15.

La FREQUENZA DI TAGLIO, che può variare tra circa 30 Hz e 12 KHz si specifica al SID come un numero a 11 bit (0-2047), quindi la variazione di unità corrisponde a una differenza della frequenza di taglio di circa 6 Hz, sufficientemente piccola per la maggior parte delle applicazioni.
Il numero a 11 bit va spezzato in modo da porre i 3 bit meno significativi nel byte 54293 (registro numero 21, il 22-esimo) e gli 8 bit più significativi nel byte 54294 (registro numero 22, il 23-esimo), come indicato nella Figura 3.16.

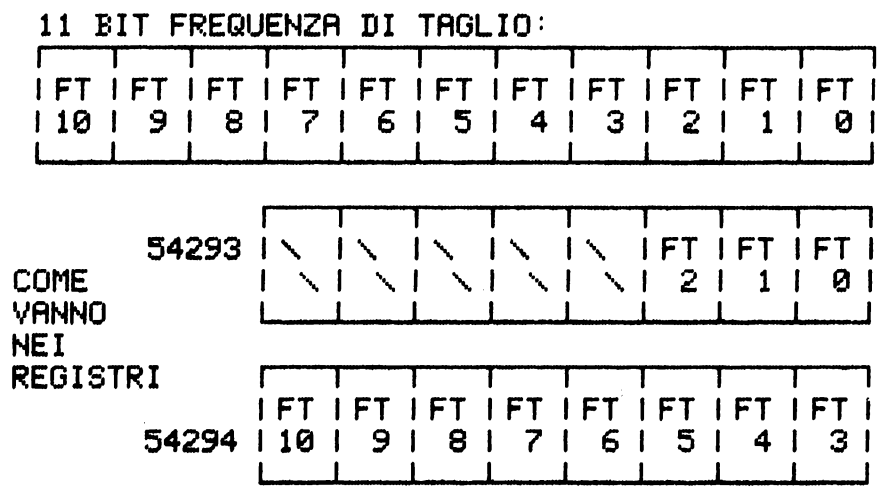


Figura 3.16 Frequenza di taglio

In pratica se FT è la frequenza di taglio si devono eseguire le due POKE indicate:

POKE 54293, FT AND 7
POKE 54294, INT(FT/8)

Non cercare di rileggere i valori posti nei byte, dato che non li otteresti uguali a quelli scritti poichè i registri del SID sono a sola scrittura.

Come per il volume, anche il filtro è comune a tutte 3 le voci. E' perciò impossibile selezionare un filtro diverso per ogni voce, mentre è possibile scegliere quali voci devono essere filtrate e quali no.
Per selezionare un determinato tipo di filtro si deve porre a 1 il bit corrispondente nel registro numero 24 (il 25-esimo) del SID, all'indirizzo 54296 (lo stesso del volume). Osserva la Figura 3.17.

```
|        UTILIZZO BIT REGISTRO 54296          |
|  7  |  6  |  5  |  4  |  3   |  2   |  1   |  0   |
|     | HP  | BP  | LP  | VOL3 | VOL2 | VOL1 | VOL0 |

        |     |     |
        |     |     |
        |     |     ---->FILTRO PASSA BASSO
        |     ------------>FILTRO PASSA BANDA
        ------------------>FILTRO PASSA ALTO
BIT=1 PER ATTIVO
BIT=0 PER NON ATTIVO
```

Figura 3.17 Utilizzo registro di indirizzo 54296

Poichè anche questo registro non può essere riletto, per selezionare i vari filtri, se V è il volume (da 0 a 15), si devono usare le istruzioni:

POKE 54296, V OR 16	per attivare il filtro passa basso
POKE 54296, V OR 32	per attivare il filtro passa banda
POKE 54296, V OR 64	per attivare il filtro passa alto
POKE 54296, V OR 80	per attivare il filtro a tacca (80=16+64)

Per indicare, invece, quali voci vanno filtrate, si utilizzano i bit meno significativi del registro numero 23, cioè del byte di indirizzo 54295, come indicato nella Figura 3.18.

```
UTILIZZO BIT REGISTRO 54295

| 7 | 6 | 5 | 4 |   3   |  2  |  1  |  0  |
|   |   |   |   | FVE   | FV3 | FV1 | FV0 |

                  |     |     |     |
                  |     |     |     ->FILTRO VOCE1
                  |     |     ------->FILTRO VOCE2
                  |     ------------->FILTRO VOCE3
                  ------------------->FILTRO SEGNALE EST.

BIT=1 PER ATTIVO
BIT=0 PER NON ATTIVO
```

Figura 3.18 Utilizzo registro di indirizzo 54295

Il quarto bit di questo stesso registro indica se anche il segnale esterno, che è connesso al SID, deve essere filtrato. Infatti il SID è in grado di ricevere un segnale dall'esterno e fonderlo con quelli delle 3 voci, permettendo effetti polifonici collegando tra loro le uscite di più SID oppure il segnale da un HI-FI.
A noi interessa relativamente; normalmente possiamo trascurare i 4 bit più significativi del registro numero 23 (il 24-esimo) e usare l'istruzione:

POKE 54295, FV

dove per FV vale quanto esposto nella Tabella 3.5. Se esiste un segnale esterno, per ottenere il filtraggio è sufficiente aggiungere 8 al valore di FV.

VALORE FV	VOCI FILTRATE 1	2	3
0	NO	NO	NO
1	SI	NO	NO
2	NO	SI	NO
3	SI	SI	NO
4	NO	NO	SI
5	SI	NO	SI
6	NO	SI	SI
7	SI	SI	SI

Tabella 3.5 Valori per registro 54295

Il SID ci permette però di aggiungere un altro effetto, legato alla frequenza di taglio indicata per i filtri.

3.3.1 La risonanza

La risonanza è un'esaltazione, un aumento di ampiezza, delle frequenze prossime alla frequenza di taglio. Ad esempio, per un filtro passa alto, l'andamento dell'ampiezza delle frequenze sarebbe quella indicata nella Figura 3.19.

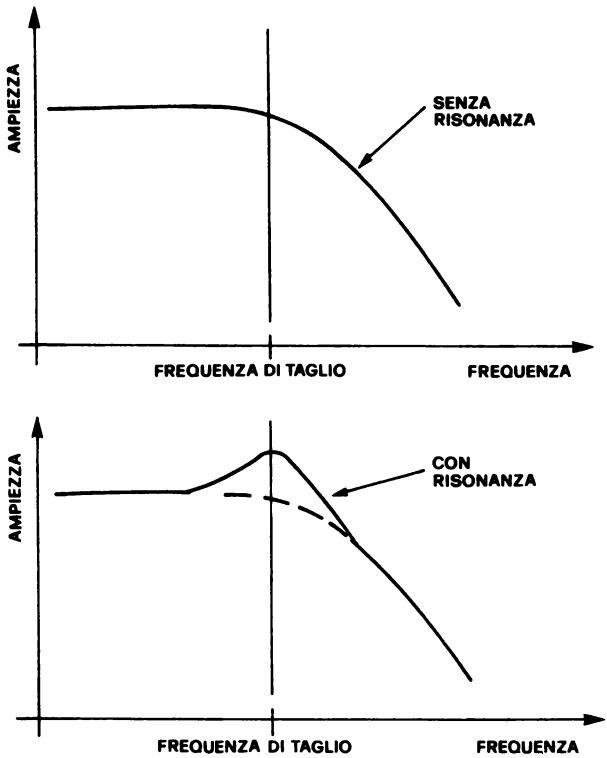


Figura 3.19 Andamento ampiezza frequenza per filtro passa alto

Questo effetto permette di ottenere suoni più decisi, privilegiando la frequenza di taglio sulle altre frequenze. Con il SID puoi graduare l'effetto di risonanza. Per ottenerlo, infatti, devi porre un valore nei 4 bit più significativi del registro numero 23 (il 24-esimo), appena visto per l'uso dei filtri.
Tale valore sarà compreso tra 0 (niente risonanza) e 15 (massima risonanza) e andrà scritto nel byte 54295 con:

POKE 54295, R*8+FV
dove:

R=0...15 è il valore per la risonanza
FV=0...15 è il numero che indica le voci da filtrare.

Osserva la Figura 3.20.

SIGNIFICATO BIT REGISTRO 54295

7	6	5	4	3	2	1	0
RI3	RI2	RI1	RI0	FVE	FV3	FV2	FV1
RISONANZA				VOCI DA FILTRARE			

Figura 3.20 Utilizzo registro di indirizzo 54295

3.4 ANCORA DI PIU'

Nell'esaminare le capacità sonore del SID abbiamo visto come usare i diversi registri, ma se ci fai caso vedi che sono rimasti dei buchi. La mappa, riportata nella Figura 3.21, ti indica quali registri e quali bit abbiamo già esaminato.

REG. SID	IND. DEC.	SIGNIFICATO BIT 7	6	5	4	3	2	1	0
0	54272	F7	F6	F5	F4	F3	F2	F1	F0
1	54273	F15	F14	F13	F12	F11	F10	F9	F8
2	54274	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0
3	54275	--	--	--	--	PW11	PW10	PW9	PW8
4	54276	RB	OI	ODS	OT	●	●	●	GATE
5	54277	A3	A2	A1	A0	D3	D2	D1	D0
6	54278	S3	S2	S1	S0	R3	R2	R1	R0
7	54279	F7	F6	F5	F4	F3	F2	F1	F0
8	54280	F15	F14	F13	F12	F11	F10	F9	F8
9	54281	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0
10	54282	--	--	--	--	PW11	PW10	PW9	PW8
11	54283	RB	OI	ODS	OT	●	●	●	GATE
12	54284	A3	A2	A1	A0	D3	D2	D1	D0
13	54285	S3	S2	S1	S0	R3	R2	R1	R0
14	54286	F7	F6	F5	F4	F3	F2	F1	F0

15	54287	F15	F14	F13	F12	F11	F10	F9	F8
16	54288	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0
17	54289	--	--	--	--	PW11	PW10	PW9	PW8
18	54290	RB	OI	ODS	OT	●	●	●	GATE
19	54291	A3	A2	A1	A0	D3	D2	D1	D0
20	54292	S3	S2	S1	S0	R3	R2	R1	R0
21	54293	--	--	--	--	--	FT2	FT1	FT0
22	54294	FT10	FT9	FT8	FT7	FT6	FT5	FT4	FT3
23	54295	RI3	RI2	RI1	RI0	FVE	FV3	FV2	FV1
24	54296	●	HP	BP	LP	VOL3	VOL2	VOL1	VOL0
25	54297	0	0	0	0	0	0	0	0
26	54298	0	0	0	0	0	0	0	0
27	54299	●	●	●	●	●	●	●	●
28	54300	●	●	●	●	●	●	●	●

NOTE:

F PER FREQUENZA SUONO
PW PER PARTE POSITIVA IMPULSO
RB PER RUMORE BIANCO
OI PER ONDA IMPULSIVA
ODS PER ONDA DENTE DI SEGA
OT PER ONDA TRIANGOLARE
A PER ATTACK
D PER DECAY
S PER SUSTAIN
R PER RELEASE
FT PER FREQUENZA DI TAGLIO
RI PER RISONANZA
FV PER FILTRO
FVE PER FILTRO SEGNALE ESTERNO
HP PER FILTRO PASSA ALTO
BP PER FILTRO PASSA BANDA
LP PER FILTRO PASSA BASSO
VOL PER VOLUME

Figura 3.21 Mappa dei registri del SID

I tondini pieni indicano bit e byte di cui non abbiamo visto l'utilizzo e di cui ci occupiamo ora.

3.4.1 Modulazione ad anello e sincronizzazione

I registri numero 4, 11 e 18 (rispettivamente quinto, 12-esimo e 19-esimo, di indirizzi 54276, 54283 e 54290) oltre a permettere di selezionare la forma d'onda e attivare il generatore di inviluppo per ogni voce (vedi Paragrafo 3.1.2), contengono altri 3 bit, come indicato nella Figura 3.22.

7	6	5	4	3	2	1	0
RB	OI	ODS	OT	TEST	RING MOD.	SYNC	GATE

Figura 3.22 Significato bit registri numero 4, 11 e 18

RING MODULATION. Quando il bit 2 dei registri indicati è posto a 1 e contemporaneamente è selezionata l'onda triangolare, l'uscita della voce a cui è associato il registro è sostituita dalla modulazione ad anello (RING MOD.):
registro 4, voce 1 modulata con voce 3
registro 11, voce 2 modulata con voce 1
registro 18, voce 3 modulata con voce 2
le frequenze devono essere diverse da zero.
La modulazione ad anello permette di variare ulteriormente la struttura armonica dei suoni generati, aggiungendo frequenze non armoniche e permettendo ulteriori mutamenti variando la frequenza del generatore associato. Ma è più facile capirlo provando che leggendo le spiegazioni. Si rivela molto utile per ottenere effetti tipo gong o altri suoni metallici.

SYNC. Quando si pone a 1 il bit 1 dei 3 registri indicati si ottiene la sincronizzazione tra la frequenza fondamentale della voce del registro e la frequenza fondamentale di un'altra voce (valgono le stesse relazioni della modulazione ad anello), che deve essere diversa da zero e possibilmente minore della prima. Anche in questo caso la variazione (meglio se in tempo reale) dei valori della frequenza della voce associata permette di ottenere strutture armoniche più complesse di quelle di partenza.
La sincronizzazione, in pratica, è lo AND logico tra le due frequenze di base, per cui la risultante è zero quando una delle due è zero.

TEST. Porre a 1 questo bit BLOCCA e RESETTA (pone a zero i registri) la voce che è controllata da questo registro. Il che vuol dire che cessano tutte le funzioni di tale generatore sonoro, fino a quando lo stesso bit non viene riportato a zero. Può essere comodo usato in un programma in quanto è un metodo più veloce che fare 24 POKE nell'area dei registri. Utile anche, nella programmazione in linguaggio macchina, per sincronizzare il SID (una voce del SID) con un evento esterno.

3.4.2 Cambiamenti dinamici del suono

Nel registro numero 24 (il 25-esimo, volume e filtri) c'è un bit che non abbiamo mai usato, quello di posizione 7. Osserva la Figura 3.23.

SIGNIFICATO BIT REGISTRO 54296

7	6	5	4	3	2	1	0
3 OFF	HP	BP	LP	VOL 3	VOL 2	VOL 1	VOL 0

Figura 3.23 Bit di posizione 7 del registro 54296

3 OFF: Porre 1 in questo bit produce l'effetto di eliminare qualunque uscita dal generatore numero 3, impedire cioè che la voce 3 possa emettere suoni. Quale la sua funzione pratica?
Abbiamo visto che il SID è mappato in memoria e che si accede ai suoi registri con delle POKE. Dei suoi 29 registri 25 sono A SOLA SCRITTURA (WRITE ONLY), cioè tentativi di lettura, per motivi di mancanza di connessioni, non restituiscono i valori realmente contenuti nei registri, e, a questo punto, li abbiamo già esaminati tutti. Restano 4 registri, che SONO A SOLA LETTURA. I primi 2 permettono di leggere (digitalizzato) lo stato di due potenziometri (numero 25, il 26-esimo di indirizzo 54297, e numero 26, il 27-esimo di indirizzo 54298), tipicamente due paddle, ma non solo quello.

Il registro numero 27 (il 28-esimo di indirizzo 54299), indicato nelle mappe come OSC3/RANDOM, riflette in ogni istante l'andamento dell'uscita audio della voce 3, a seconda della forma d'onda selezionata per tale voce:

• se hai selezionato l'onda triangolare, il contenuto del registro numero 27 varia da 0 a 255, poi scende a 0 ciclicamente, con una velocità che dipende SOLO dalla frequenza scelta;

• se hai selezionato l'onda a dente di sega, il valore varia da 0 a 255 e torna immediatamente a 0 per riprendere a salire;

• se hai selezionato l'onda impulsiva ottieni una sequenza di 0 e 255 corrispondenti con le fasi negative e positive dell'onda;

• la selezione del rumore bianco fa comparire nel registro numeri a caso tra o e 255.

In tutti i casi l'inviluppo scelto per la voce 3 non influenzerà i risultati delle letture, inoltre la frequenza selezionata per questa voce deve essere diversa da zero.
Il registro numero 28 (il 29-esimo di indirizzo 54300), invece, riflette proprio i mutamenti nell'ampiezza dell'uscita audio della voce 3, cioè dell'andamento ADSR.
Entrambi i registri (con i loro contenuti) possono essere utilizzati per creare effetti speciali come il TREMOLO, lo WAH-WAH, le sirene, e altri ancora, fornendo un ulteriore modo per modificare i suoni prodotti in base dalle voci del SID.
Poichè spesso l'uscita audio della voce 3 sarà indesiderata, in quanto sarà stata programmata (la voce 3) per leggere i valori dai due registri appena visti è stato reso possibile INIBIRNE l'uscita, con una semplice POKE nel registro numero 24, che ponga a 1 il bit di posizione 6.

3.5 PER CONCLUDERE

Abbiamo visto praticamente tutte le possibilità del SID, che, usate con un pò di fantasia, permettono la generazione di una varietà notevolissima di suoni. Per finire ti proponiamo ancora qualche esempio in cui ritroverai utilizzate le informazioni viste nei paragrafi precedenti.

Il programma SUONO1 è una dimostrazione delle capacità AVANZATE del SID: filtri, modulazione ad anello, sincronizzazione e variazione dinamica del suono.

```
1 REM SUONO1
10 POKE53280,0:POKE53281,0
15 PRINTCHR$(154)CHR$(147)
20 V1=54272:V3=54286
30 PRINT"ONDA TRIANGOLARE"
40 POKEV1,0
50 POKEV1+1,22
60 POKEV1+5,0
```

```
70 POKEV1+6,240
80 POKEV1+4,17
90 POKEV1+24,15
100 FORI=1TO2000:NEXT
130 PRINT:PRINT"SINCRONIZZATA"
150 POKEV3,0
160 POKEV3+1,10
170 POKEV3+4,16
200 POKEV1+4,19
210 FORI=1TO2000:NEXT
220 PRINT:PRINT"E MODULATA"
230 POKEV1+4,21
240 FORI=1TO2000:NEXT
250 PRINT:PRINT"TREMOLO"
260 POKEV1+4,17
280 POKEV3,60:POKEV3+1,0
300 FORI=1TO300:POKEV1+24,PEEK(54299)/16:NEXT
400 PRINT:PRINT"FILTRO PASSA BASSO"
410 POKEV1+24,31:POKEV1+23,1:POKEV1+22,19
420 FORI=1TO2000:NEXT
430 PRINT:PRINT"FILTRO PASSA BANDA"
440 POKEV1+24,47
450 FORI=1TO2000:NEXT
460 PRINT:PRINT"FILTRO PASSA ALTO"
470 POKEV1+24,79
480 FORI=1TO2000:NEXT
481 PRINT:PRINT"FILTRO A TACCA"
482 POKEV1+24,95
483 FORI=1TO2000:NEXT
490 PRINT:PRINT"VIBRATO"
500 POKEV3,22:POKEV1+24,47
510 FORI=1TO500:POKEV1+22,PEEK(54299)/2:NEXT
520 FORI=0TO24:POKEV1+I,0:NEXT
```

COMMENTO A SUONO1.
Linea 10: inizializza il video.
Linee 20-100: produce un'onda triangolare.
Linee 130-210: sincronizza la voce 1 con la voce 3, ponendo nel registro di controllo della voce 1 (54276 (D404H)) il bit di posizione 1 a 1.
Linee 220-240: modula la voce 1 con la voce 3, ponendo nel registro di controllo della voce 1 (54276 (D404H)) il bit di posizione 2 a 1.

Linee 250-300: ottiene un effetto di tremolo variando dinamicamente il suono. Pone cioè nel registro del volume (54296 (D418H)) il contenuto del registro 54299 (D41BH) che contiene il valore digitale dell'uscita della voce 3. Tale valore (che varia da 0 a 255) è stato diviso per 16 poichè il volume può variare solo tra 0 e 15.
Linee 400-420: aziona il filtro passa basso, ponendo:

a 1 il bit di posizione 4 del registro 54296 (D418H) per selezionare il filtro passa basso;

a 1 il bit di posizione 0 del registro 54295 (D417H) per filtrare la voce 1;

19 nel registro 54294 (D416H) per ottenere come frequenza di taglio la stessa frequenza a cui oscilla la voce 1.

Linee 430-450: aziona il filtro passa banda ponendo a 1 il bit di posizione 5 del registro 54296 (D418H) per selezionare tale filtro.
Linee 460-480: aziona il filtro passa alto ponendo a 1 il bit di posizione 6 del registro 54296 (D418H) per selezionare tale filtro.
Linee 481-483: aziona il filtro a tacca ponendo a 1 i bit di posizione 4 e 6 del registro 54296 (D418H) per selezionare i filtri passa basso e passa alto.
Linee 490-520: ottiene un effetto vibrato variando dinamicamente il suono come per l'effetto tremolo, ma agendo sulla frequenza di taglio del filtro passa banda, anzichè sul volume.
Il programma SUONO2 dimostra come si possa usare completamente la voce 3 (sia oscillatore che generatore di inviluppo) per ottenere una variazione dinamica del suono. La frequenza di uscita del generatore 1 è proporzionale all'uscita che avrebbe la voce 3 se non fosse disabilitata dal bit di posizione 7 del registro 54296 (D418H).

```
1 REM SUONO2
5 REM CAMBIAMENTO DINAMICO CON VOCE 3
10 V1=54272:V3=54286
20 POKEV3,20
30 POKEV3+1,0
40 POKEV3+5,240
50 POKEV3+6,253
60 POKEV1+24,15+128
70 POKEV1,0
80 POKEV1+5,0:POKEV1+6,240
90 POKEV1+4,33
100 POKEV3+4,17
110 FORI=0TO400
115 POKEV1+1,PEEK(V1+27)/255*PEEK(V1+28):NEXT
120 POKEV3+4,16
130 FORI=0TO500
135 POKEV1+1,PEEK(V1+27)/255*PEEK(V1+28):NEXT
```

COMMENTO A SUONO2.
Linee 20-50: seleziona per la voce 3 una bassa frequenza di oscillazione, attacco=15, decadimento=0, sostenimento=15, rilascio=13.
Linea 60: pone il volume a 15 e disabilita la voce 3.
Linee 70-90: seleziona per la voce 1: attacco=0, decadimento=0, sostenimento=15, rilascio=0. Pone a 0 il byte meno significativo della frequenza, seleziona la forma d'onda a dente di sega e fa partire l'oscillatore.
Linea 100: seleziona per la voce 3 la forma d'onda triangolare e fa partire l'oscillatore.
Linee 110-115: pone nel byte più significativo della frequenza della voce 1 il valore che avrebbe l'uscita della voce 3 se non fosse disabilitata.
Linee 130-135: fa partire la fase di rilascio della voce 3 e continua con il calcolo della frequenza della voce 1 per 500 cicli.

Il programma SUONO3 dimostra come si possa ottenere una variazione dinamica del suono senza appoggiarsi alla voce 3, ma, per esempio, a ciò che accade sul video. In questo caso viene visualizzata una pallina che rimbalza; la frequenza del suono emesso dal SID è proporzionale all'altezza della pallina e il volume alla posizione orizzontale della pallina.

```
1 REM SUONO3
5 REM CAMBIAMENTO DINAMICO SENZA VOCE 3
100 REM  123456789012345678901234
110 DATA"                        "
120 DATA"                        "
130 DATA"          ******        "
140 DATA"        **********      "
150 DATA"       ************     "
160 DATA"      **************    "
170 DATA"     ****************   "
180 DATA"     ****************   "
190 DATA"    ******************  "
200 DATA"    ******************  "
210 DATA"    ******************  "
220 DATA"    ******************  "
230 DATA"    ******************  "
240 DATA"     ****************   "
250 DATA"     ****************   "
260 DATA"      **************    "
270 DATA"       ************     "
280 DATA"        **********      "
290 DATA"          ******        "
```

```
300 DATA"                                  "
310 DATA"                                  "
320 FORI=1TO21:READA$:PRINTA$
330 FORJ=0TO2:B$=MID$(A$,J*8+1,8)
340 BY=0:FORK=1TO8
350 IFMID$(B$,K,1)="*"THENBY=BY+2↑(8-K)
360 NEXT:POKE896+I*3+J,BY:NEXT:NEXT
370 POKE2040,14:POKE53269,1
380 POKE53248,10:PRINTCHR$(147)
390 V1=54272:V2=54279
400 POKEV1,0:POKEV1+1,0:POKEV2,0:POKEV2+1,10
410 POKEV1+5,0:POKEV1+6,240
415 POKEV2+5,8:POKEV2+6,0:POKEV2+3,8
420 POKEV1+4,33
430 POKEV1+24,15
440 G=2000
450 T=T+.03
460 YN=G*T*T+V0*T+S0:VN=(YN-Y)/.03:Y=YN
470 IFV<0ANDVN>0ANDY>231THENPOKEV1+1,0:END
480 V=VN
490 IFY>230THENRB=-1:Y=230
500 POKEV1+1,200-Y/3
510 POKE53249,Y:POKE53248,PEEK(53248)+2
515 POKEV1+24,17-PEEK(53248)/16
520 IFRBTHENV=-SQR(.7*V*V):V0=V:S0=230:T=0
530 IFRBTHENRB=0:POKEV2+4,64:POKEV2+4,65
540 GOTO450
```

COMMENTO A SUONO3.

Linee 0-380: crea lo sprite per la pallina.

Linee 390-430: inizializza le due voci che verranno usate.

Linea 440: inizializza la costante G, che rappresenta la gravità per la simulazione della pallina.

Linea 450: inizia un ciclo il cui indice è la variabile T; essa verrà inizializzata nuovamente quando sarà verificata la condizione di linea 490, cioè ogni volta che la pallina toccherà il pavimento.

Linea 460: calcola, in funzione della gravità, della velocità iniziale e del valore iniziale della Y, il valore della Y e della velocità attuale.

Linea 470: calcola la condizione in cui la pallina è ferma ed eventualmente esce.

Linea 490: se è verificata una condizione di rimbalzo, allora la variabile booleana RB viene posta a VERO.

Linea 500: la frequenza della voce 1 è proporzionale alla quota della pallina.
Linea 510: aggiorna la posizione dello sprite.
Linea 520: il volume è proporzionale alla posizione orizzontale della pallina.
Linee 520-530: se è verificata la condizione di rimbalzo, allora la velocità cambia di segno e viene ridotta in modulo (la pallina non è perfettamente elastica), vengono ripristinate le condizioni iniziali e la voce 2 ha il compito di produrre il rumore della pallina che colpisce il pavimento.
Linea 540: chiude il ciclo.

Seguono i programmi SUONO5, SUONO6, SUONO7, SUONO8, SUONO9 e SUONO10, che producono effetti sonori speciali.

```
1 REM SUONO5
5 REM MANIA DEL CALCOLATORE
10 SI=54272
20 POKESI+6,240:POKESI+5,0
25 POKESI+3,8:POKESI+4,65
30 POKESI+24,15
40 FORL=1TO100
50 POKESI+1,RND(1)*256
60 FORM=1TO50:NEXT
70 NEXT
80 POKESI+1,0:POKESI+24,0
```

```
1 REM SUONO6
1000 REM SIRENA
1010 VV=54272:POKEVV+6,0:POKEVV+5,237
1015 POKEVV+24,15
1020 POKEVV+2,200:POKEVV+3,0
1025 AA=55:BB=42
1030 POKEVV+1,60:POKEVV+4,65
1040 FOR NN=1TO12
1050 POKEVV+1,AA
1060 FORMM=1TO300:NEXTMM
1070 POKEVV+1,BB
1080 FORMM=1TO300:NEXTMM
1085 IF NN=6THEN AA=53:BB=40
1090 NEXT NN
1100 POKEVV+4,0
1120 POKEVV+24,0
```

```
1 REM SUONO7
5 REM ALLARME ROSSO
10 SI=54272
20 POKESI+6,240:POKESI+5,0
25 POKESI+3,8:POKESI+4,65
30 POKESI+24,15
40 FORL=1TO10
50 FORM=10TO50STEP2
60 POKESI+1,M
70 FORN=1TO10
80 NEXTN
90 NEXTM
100 POKESI+1,0
110 FORM=1TO100
120 NEXTM
130 NEXTL
140 POKESI+24,0
```

```
1 REM SUONO8
5 REM TELEFONO
10 SI=54272
20 POKESI+6,240:POKESI+5,0
25 POKESI+3,7:POKESI+4,65
30 POKESI+24,15
40 FORL=1TO5
50 FORM=1TO50
60 POKESI+1,47
70 FORN=1TO5
80 NEXTN
90 POKESI+1,0
100 NEXTM
110 FORM=1TO3000
120 NEXTM
130 NEXTL
140 POKESI+24,0
```

```
1 REM SUONO9
5 REM ONDE DEL MARE
10 SI=54272
20 POKESI+6,240:POKESI+5,0
25 POKESI+3,7:POKESI+4,129
30 POKESI+1,40
40 FORL=1TO10
50 D=INT(RND(1)*5)*50+50
60 FORM=3TO15
70 POKESI+24,M
80 FORN=1TOD
90 NEXTN
100 NEXTM
110 FORM=15TO3STEP-1
120 POKESI+24,M
130 FORN=1TOD
140 NEXTN
150 NEXTM
160 NEXTL
170 POKESI+1,0
180 POKESI+24,0
```

```
1 REM SUONO10
5 REM CINGUETTIO
10 SI=54272
20 POKESI+6,240:POKESI+5,0
25 POKESI+3,8:POKESI+4,65
30 POKESI+24,15
40 FORL=1TO20
50 FORM=210TO180+INT(RND(1)*20)STEP-2
60 POKESI+1,M
70 NEXTM
80 FORM=0TOINT(RND(1)*80)+80
90 NEXTM
100 NEXTL
110 POKESI+1,0
120 POKESI+24,0
```

3.6 VALORI DELLE NOTE

Riportiamo i valori da porre nel byte più significativo (HI) e nel byte meno significativo (LO) del registro della frequenza per ottenere una nota musicale per 8 ottave.

TABELLA DELLE NOTE

NOTA	FREQ.	VAL.	HI	LOW
DO-0	16.4	278	1	22
DO#-0	17.3	295	1	39
RE-0	18.4	313	1	57
RE#-0	19.4	331	1	75
MI-0	20.6	351	1	95
FA-0	21.8	372	1	116
FA#-0	23.1	394	1	138
SOL-0	24.5	417	1	161
SOL#-0	26	442	1	186
LA-0	27.5	468	1	212
LA#-0	29.1	496	1	240
SI-0	30.9	526	2	14
DO-1	32.7	557	2	45
DO#-1	34.6	590	2	78
RE-1	36.7	625	2	113
RE#-1	38.9	662	2	150
MI-1	41.2	702	2	190
FA-1	43.7	743	2	231
FA#-1	46.2	788	3	20
SOL-1	49	834	3	66
SOL#-1	51.9	884	3	116
LA-1	55	937	3	169
LA#-1	58.3	992	3	224

NOTA	FREQ.	VAL.	HI	LOW
SI-1	61.7	1051	4	27
DO-2	65.4	1114	4	90
DO#-2	69.3	1180	4	156
RE-2	73.4	1250	4	226
RE#-2	77.8	1324	5	44
MI-2	82.4	1403	5	123
FA-2	87.3	1487	5	207
FA#-2	92.5	1575	6	39
SOL-2	98	1669	6	133
SOL#-2	103.8	1768	6	232
LA-2	110	1873	7	81
LA#-2	116.5	1985	7	193
SI-2	123.5	2103	8	55
DO-3	130.8	2228	8	180
DO#-3	138.6	2360	9	56
RE-3	146.8	2500	9	196
RE#-3	155.6	2649	10	89
MI-3	164.8	2807	10	247
FA-3	174.6	2973	11	157
FA#-3	185	3150	12	78
SOL-3	196	3338	13	10
SOL#-3	207.7	3536	13	208
LA-3	220	3746	14	162
LA#-3	233.1	3969	15	129
SI-3	246.9	4205	16	109
DO-4	261.6	4455	17	103
DO#-4	277.2	4720	18	112
RE-4	293.7	5001	19	137
RE#-4	311.1	5298	20	178
MI-4	329.6	5613	21	237
FA-4	349.2	5947	23	59
FA#-4	370	6300	24	156
SOL-4	392	6675	26	19
SOL#-4	415.3	7072	27	160
LA-4	440	7493	29	69
LA#-4	466.2	7938	31	2
SI-4	493.9	8410	32	218
DO-5	523.3	8910	34	206
DO#-5	554.4	9440	36	224
RE-5	587.3	10001	39	17
RE#-5	622.3	10596	41	100

NOTA	FREQ.	VAL.	HI	LOW
MI-5	659.3	11226	43	218
FA-5	698.5	11894	46	118
FA#-5	740	12601	49	57
SOL-5	784	13350	52	38
SOL#-5	830.6	14144	55	64
LA-5	880	14985	58	137
LA#-5	932.3	15876	62	4
SI-5	987.8	16820	65	180
DO-6	1046.5	17820	69	156
DO#-6	1108.7	18880	73	192
RE-6	1174.7	20003	78	35
RE#-6	1244.5	21192	82	200
MI-6	1318.5	22452	87	180
FA-6	1396.9	23787	92	235
FA#-6	1480	25202	98	114
SOL-6	1568	26700	104	76
SOL#-6	1661.2	28288	110	128
LA-6	1760	29970	117	18
LA#-6	1864.7	31752	124	8
SI-6	1975.5	33640	131	104
DO-7	2093	35641	139	57
DO#-7	2217.5	37760	147	128
RE-7	2349.3	40005	156	69
RE#-7	2489	42384	165	144
MI-7	2637	44904	175	104
FA-7	2793.8	47574	185	214
FA#-7	2960	50403	196	227
SOL-7	3136	53400	208	152
SOL#-7	3322.4	56576	221	0
LA-7	3520	59940	234	36
LA#-7	3729.3	63504	248	16

CAPITOLO 4

I REGISTRI DEL VIC II E I REGISTRI DEL SID

4.1 I REGISTRI DEL VIC II

Riportiamo l'elenco dei registri del VIC II, indicando per ogni registro il numero, l'indirizzo decimale e esadecimale, e la funzione.

```
R E G I S T R I   D E L   V I C   I I

REG. INDIRIZZO      FUNZIONE

0    53248(D000H)   POS. X SPRITE NUM.0
1    53249(D001H)   POS. Y SPRITE NUM.0
2    53250(D002H)   POS. X SPRITE NUM.1
3    53251(D003H)   POS. Y SPRITE NUM.1
4    53252(D004H)   POS. X SPRITE NUM.2
5    53253(D005H)   POS. Y SPRITE NUM.2
6    53254(D006H)   POS. X SPRITE NUM.3
7    53255(D007H)   POS. Y SPRITE NUM.3
8    53256(D008H)   POS. X SPRITE NUM.4
9    53257(D009H)   POS. Y SPRITE NUM.4
10   53258(D00AH)   POS. X SPRITE NUM.5
11   53259(D00BH)   POS. Y SPRITE NUM.5
12   53260(D00CH)   POS. X SPRITE NUM.6
13   53261(D00DH)   POS. Y SPRITE NUM.6
14   53262(D00EH)   POS. X SPRITE NUM.7
15   53263(D00FH)   POS. Y SPRITE NUM.7
16   53264(D010H)
           BIT 7: MSB X SPRITE NUM.7
           BIT 6: MSB X SPRITE NUM.6
           BIT 5: MSB X SPRITE NUM.5
           BIT 4: MSB X SPRITE NUM.4
           BIT 3: MSB X SPRITE NUM.3
           BIT 2: MSB X SPRITE NUM.2
           BIT 1: MSB X SPRITE NUM.1
           BIT 0: MSB X SPRITE NUM.0
17   53265(D011H) REGISTRO DI CONTROLLO
           BIT 7: MSB REGISTRO DI LINEA
           BIT 6: 1 MODO SFONDO PROGRAMMABILE
```

```
 REG. INDIRIZZO     FUNZIONE

          BIT 5: 1 MODO PAGINA GRAFICA
          BIT 4: 0 ANNULLAMENTO SCHERMO
          BIT 3: 0 SCHERMO A 24 RIGHE
          BIT 2-0: POS. Y DELLE SCRITTE
                   (SCORRIMENTO FINE)
18  53266(D012H) REGISTRO DI LINEA
19  53267(D013H) POS. X LIGHT-PEN
20  53268(D014H) POS. Y LIGHT-PEN
21  53269(D015H) ABILITAZIONE SPRITE
          BIT 7: 1 SPRITE NUM.7
          BIT 6: 1 SPRITE NUM.6
          BIT 5: 1 SPRITE NUM.5
          BIT 4: 1 SPRITE NUM.4
          BIT 3: 1 SPRITE NUM.3
          BIT 2: 1 SPRITE NUM.2
          BIT 1: 1 SPRITE NUM.1
          BIT 0: 1 SPRITE NUM.0
22  53270(D016H) REGISTRO DI CONTROLLO
          BIT 7-5: NON USATI
          BIT 4: 1 MODO MULTICOLORE
          BIT 3: 0 SCHERMO A 38 COLONNE
          BIT 2-0: POS. X DELLE SCRITTE
                   (SCORRIMENTO FINE)
23  53271(D017H) ESPANSIONE VERTICALE SPRITE
          BIT 7: 1 SPRITE NUM.7
          BIT 6: 1 SPRITE NUM.6
          BIT 5: 1 SPRITE NUM.5
          BIT 4: 1 SPRITE NUM.4
          BIT 3: 1 SPRITE NUM.3
          BIT 2: 1 SPRITE NUM.2
          BIT 1: 1 SPRITE NUM.1
          BIT 0: 1 SPRITE NUM.0
24  53272(D018H) REGISTRO CONTROLLO MEMORIA
          BIT 7-4: IND. BASE MAPPA VIDEO
          BIT 3-1: IND. BASE MAPPA CARATTERI
          BIT 0: NON USATO
25  53273(D019H) REGISTRO DI STATO INTERRUPT
          BIT 7: 1 INTERRUPT LANCIATO
                   DAL VIC II
```

```
REG.INDIRIZZO     FUNZIONE
          BIT 6-4: NON USATI
          BIT 3: 1 INTERRUPT LANCIATO
                   DALLA LIGHT-PEN
          BIT 2: 1 INTERRUPT COLLISIONE
                   TRA SPRITE
          BIT 1: 1 INTERRUPT COLLISIONE
                   SPRITE-SFONDO
          BIT 0: 1 INTERRUPT REGISTRO DI LINEA
26  53274(D01AH) REGISTRO ABILITAZIONE
                 INTERRUPT: 1=ABILITATO
          (UTILIZZO DEI BIT ANALOGO AL
           REGISTRO PRECEDENTE)
27  53275(D01BH) PRIORITA' SPRITE-SFONDO
          (1-SFONDO, 0-SPRITE)
          BIT 7: SPRITE NUM.7
          BIT 6: SPRITE NUM.6
          BIT 5: SPRITE NUM.5
          BIT 4: SPRITE NUM.4
          BIT 3: SPRITE NUM.3
          BIT 2: SPRITE NUM.2
          BIT 1: SPRITE NUM.1
          BIT 0: SPRITE NUM.0
28  53276(D01CH) SPRITE MULTICOLORE
          BIT 7: 1 SPRITE NUM.7
          BIT 6: 1 SPRITE NUM.6
          BIT 5: 1 SPRITE NUM.5
          BIT 4: 1 SPRITE NUM.4
          BIT 3: 1 SPRITE NUM.3
          BIT 2: 1 SPRITE NUM.2
          BIT 1: 1 SPRITE NUM.1
          BIT 0: 1 SPRITE NUM.0
29  53277(D01DH) ESPANSIONE ORIZZONTALE SPRITE
          BIT 7: 1 SPRITE NUM.7
          BIT 6: 1 SPRITE NUM.6
          BIT 5: 1 SPRITE NUM.5
          BIT 4: 1 SPRITE NUM.4
          BIT 3: 1 SPRITE NUM.3
          BIT 2: 1 SPRITE NUM.2
          BIT 1: 1 SPRITE NUM.1
          BIT 0: 1 SPRITE NUM.0
```

```
REG. INDIRIZZO      FUNZIONE

 30  53278(D01EH) COLLISIONE SPRITE-SPRITE
            BIT 7: 1 SPRITE NUM.7
            BIT 6: 1 SPRITE NUM.6
            BIT 5: 1 SPRITE NUM.5
            BIT 4: 1 SPRITE NUM.4
            BIT 3: 1 SPRITE NUM.3
            BIT 2: 1 SPRITE NUM.2
            BIT 1: 1 SPRITE NUM.1
            BIT 0: 1 SPRITE NUM.0
 31  53279(D01FH) COLLISIONE SPRITE-SFONDO
            BIT 7: 1 SPRITE NUM.7
            BIT 6: 1 SPRITE NUM.6
            BIT 5: 1 SPRITE NUM.5
            BIT 4: 1 SPRITE NUM.4
            BIT 3: 1 SPRITE NUM.3
            BIT 2: 1 SPRITE NUM.2
            BIT 1: 1 SPRITE NUM.1
            BIT 0: 1 SPRITE NUM.0
 32  53280(D020H) COLORE DEL BORDO
 33  53281(D021H) COLORE DELLO SFONDO
                 (COLORE DI SFONDO 1)
 34  53282(D022H) COLORE DI SFONDO 2
 35  53283(D023H) COLORE DI SFONDO 3
 36  53284(D024H) COLORE DI SFONDO 4
 37  53285(D025H) COLORE 0 SPRITE MULTICOLORE
 38  53286(D026H) COLORE 1 SPRITE MULTICOLORE
 39  53287(D027H) COLORE SPRITE NUM.0
 40  53288(D028H) COLORE SPRITE NUM.1
 41  53289(D029H) COLORE SPRITE NUM.2
 42  53290(D02AH) COLORE SPRITE NUM.3
 43  53291(D02BH) COLORE SPRITE NUM.4
 44  53292(D02CH) COLORE SPRITE NUM.5
 45  53293(D02DH) COLORE SPRITE NUM.6
 46  53294(D02EH) COLORE SPRITE NUM.7
```

NOTA: nella tabella MSB (Most Significative Bit) sta per BIT PIU' SIGNIFICATIVI.

4.2 I REGISTRI DEL SID

Riportiamo l'elenco dei registri del SID, indicando per ogni registro il numero, l'indirizzo decimale e esadecimale, e la funzione.

```
R E G I S T R I   D E L   S I D

REG. INDIRIZZO        FUNZIONE

0    54272(D400H)     LO FREQUENZA VOCE 1
1    54273(D401H)     HI FREQUENZA VOCE 1
2    54274(D402H)     LO LARGHEZZA ONDA
                      IMPULSIVA VOCE 1
3    54275(D403H)
           BIT 7-4:   NON USATI
           BIT 3-0:   NIBBLE PIU' SIGN. LARGHEZZA
                      ONDA IMPULSIVA VOCE 1
4    54276(D404H)     REGISTRO CONTROLLO VOCE 1
           BIT 7:  1  SELEZIONA FORMA ONDA
                      CASUALE (RUMORE)
           BIT 6:  1  SELEZIONA FORMA ONDA
                      IMPULSIVA
           BIT 5:  1  SELEZIONA FORMA ONDA
                      A DENTE DI SEGA
           BIT 4:  1  SELEZIONA FORMA ONDA
                      TRIANGOLARE
           BIT 3:  1  OSCILLATORE 1 DISABILITATO
           BIT 2:  1  OSCILL. 1 MODULATO CON
                      USCITA OSCILLATORE 3
           BIT 1:  1  OSCILL. 1 SINCRONIZZATO
                      CON FREQUENZA OSCILL. 3
           BIT 0:  1  PARTENZA ATT./DEC./SUS.
                   0  PARTENZA RELEASE
```

```
REG. INDIRIZZO        FUNZIONE

5    54277(D405H)
           BIT 7-4: SELEZIONA ATTACK GENERATORE 1
           BIT 3-0: SELEZIONA DECAY GENERATORE 1
6    54278(D406H)
           BIT 7-4: SELEZIONA SUSTAIN GENERATORE 1
           BIT 3-0: SELEZIONE RELEASE GENERATORE 1
7    54279(D407H)   LO FREQUENZA VOCE 2
8    54280(D408H)   HI FREQUENZA VOCE 2
9    54281(D409H)   LO LARGHEZZA ONDA
                    IMPULSIVA VOCE 2
10   54282(D40AH)
           BIT 7-4: NON USATI
           BIT 3-0: NIBBLE PIU' SIGN. LARGHEZZA
                    ONDA IMPULSIVA VOCE 2
11   54283(D40BH)   REGISTRO CONTROLLO VOCE 2
           BIT 7: 1 SELEZIONA FORMA ONDA
                    CASUALE (RUMORE)
           BIT 6: 1 SELEZIONA FORMA ONDA
                    IMPULSIVA
           BIT 5: 1 SELEZIONA FORMA ONDA
                    A DENTE DI SEGA
           BIT 4: 1 SELEZIONA FORMA ONDA
                    TRIANGOLARE
           BIT 3: 1 OSCILLATORE 2 DISABILITATO
           BIT 2: 1 OSCILL. 2 MODULATO CON
                    USCITA OSCILLATORE 1
           BIT 1: 1 OSCILL. 2 SINCRONIZZATO
                    CON FREQUENZA OSCILL. 1
           BIT 0: 1 PARTENZA ATT./DEC./SUS
                  0 PARTENZA RELEASE
12   54284(D40CH)
           BIT 7-4: SELEZIONA ATTACK GENERATORE 2
           BIT 3-0: SELEZIONA DECAY GENERATORE 2
13   54285(D40DH)
           BIT 7-4: SELEZIONA SUSTAIN GENERATORE 2
           BIT 3-0: SELEZIONA RELEASE GENERATORE 2
14   54286(D40EH)   LO FREQUENZA VOCE 3
15   54287(D40FH)   HI FREQUENZA VOCE 3
16   54288(D410H)   LO LARGHEZZA ONDA IMPULSIVA
                    VOCE 3
```

```
REG. INDIRIZZO        FUNZIONE
17  54289(D411H)
          BIT 7-4: NON USATI
          BIT 3-0: NIBBLE PIU' SIGN. LARGHEZZA
                   ONDA IMPULSIVA VOCE 3
18  54290(D412H)   REGISTRO CONTROLLO VOCE 3
          BIT 7: 1 SELEZIONA FORMA ONDA
                   CASUALE (RUMORE)
          BIT 6: 1 SELEZIONA FORMA ONDA
                   IMPULSIVA
          BIT 5: 1 SELEZIONA FORMA ONDA
                   A DENTE DI SEGA
          BIT 4: 1 SELEZIONA FORMA ONDA
                   TRIANGOLARE
          BIT 3: 1 OSCILLATORE 3 DISABILITATO
          BIT 2: 1 OSCILL. 3 MODULATO CON
                   USCITA OSCILLATORE 2
          BIT 1: 1 OSCILL. 3 SINCRONIZZATO
                   CON FREQUENZA OSCILL. 2
          BIT 0: 1 PARTENZA ATT./DEC./SUS
                 0 PARTENZA RELEASE
19  54291(D413H)
          BIT 7-4: SELEZIONA ATTACK GENERATORE 3
          BIT 3-0: SELEZIONA DECAY GENERATORE 3
20  54292(D414H)
          BIT 7-4: SELEZIONA SUSTAIN GENERATORE 3
          BIT 3-0: SELEZIONA RELEASE GENERATORE 3
21  54293(D415H)
          BIT 7-3: NON USATI
          BIT 2-1: LSB FREQUENZA DI TAGLIO
                   DEL FILTRO
22  54294(D416H)   HI FREQUENZA DI TAGLIO
                   DEL FILTRO
23  54295(D417H)
          BIT 7-4: RISONANZA FILTRO
          BIT 5: 1 FILTRA INPUT ESTERNO
```

```
REG. INDIRIZZO        FUNZIONE
24   54296(D418H)
           BIT 7: 0 USCITA VOCE 3 DISABILITATA
           BIT 6: 1 FILTRO PASSA ALTO
           BIT 5: 1 FILTRO PASSA BANDA
           BIT 4: 1 FILTRO PASSA BASSO
           BIT 3-0: VOLUME
25   54297(D419H)     CONVERTITORE ANALOGICO
                      -DIGITALE (PADDLE 1)
26   54298(D41AH)     CONVERTITORE ANALOGICO
                      -DIGITALE (PADDLE 2)
27   54299(D41BH)     USCITA GENERATORE 3
28   54300(D41CH)     USCITA INVILUPPO GENERATORE 3
```

NOTA: nella tabella valgono le seguenti abbreviazioni:
LO byte meno significativo di un numero formato da 2 byte;
HI byte più significativo di un numero formato da 2 byte;
MSB bit più significativi di un numero espresso in binario;
LSB bit meno significativi di un numero espresso in binario;
NIBBLE mezzo byte, cioè 4 bit.

8

INDICE

Capitolo 1 - UTILIZZO DELLA MEMORIA E CONFIGURAZIONI
1.1 Premessa 1
1.2 Variabili in memoria 7
1.3 Programma in memoria 16
1.4 Memoria di lavoro del sistema 22
1.5 Configurazioni 39
1.6 Assegnazioni memoria per I/O 44

Capitolo 2 - IL SISTEMA OPERATIVO GEOS
2.1 Il sistema operativo GEOS 49
2.2 Applicazione GEOPAINT 66
2.3 Applicazione GEOWRITER 71

CAPITOLO 1

UTILIZZO DELLA MEMORIA E CONFIGURAZIONI

1.1 PREMESSA

Abbiamo già accennato alla tecnica di gestione della memoria mediante puntatori situati nelle prime pagine della memoria RAM.

Ora ci occupiamo più dettagliatamente dei puntatori che gestiscono la parte di memoria RAM occupata dal programma in Basic e dalle sue variabili. Segue un programma che stampa sul video e, se si vuole, anche sulla stampante, il contenuto di 7 puntatori. Nel programma sono caricati con delle frasi DATA sia le stringhe di descrizione dei puntatori, che gli indirizzi dei relativi byte. Al listato del programma fa seguito il risultato su stampante.

```
1 REM PUNTBASIC
2 INPUT"RISULTATI SU STAMPANTE? (S/N) ";R$
3 PRINT:IFR$="N"THEN8
4 OPEN4,4
5 PRINT#4,"***CONTENUTO PUNTATORI A 2 BYTE***".
6 PRINT#4
8 D$="":READD$,B,A
9 IFD$="*"THEN40
10 Z=256*PEEK(A)+PEEK(B)
20 PRINTD$;SPC(2);"(";B;"/";A;")";Z
25 IFR$="N"THEN35
30 PRINT#4,D$;SPC(2);"(";B;"/";A;")";Z
31 PRINT#4
35 GOTO8
40 IFR$="S"THENCLOSE4
45 STOP
100 DATA"INIZIO PROGRAMMA BASIC",43,44
101 DATA"INIZIO VARIABILI BASIC",45,46
102 DATA"INIZIO ARRAY BASIC",47,48
103 DATA"FINE ARRAY BASIC + 1",49,50
```

```
104 DATA"IND. FONDO AREA STRINGHE",51,52
105 DATA"PUNTATORE STRINGHE",53,54
106 DATA"INDIRIZZO PIU' ALTO BASIC",55,56
107 DATA"*",0,0
```

RISULTATI PROGRAMMA

```
***CONTENUTO PUNTATORI A 2 BYTE***

INIZIO PROGRAMMA BASIC  ( 43 / 44 ) 2049

INIZIO VARIABILI BASIC  ( 45 / 46 ) 2611

INIZIO ARRAY BASIC  ( 47 / 48 ) 2646

FINE ARRAY BASIC + 1  ( 49 / 50 ) 2646

IND. FONDO AREA STRINGHE  ( 51 / 52 ) 40959

PUNTATORE STRINGHE  ( 53 / 54 ) 40960

INDIRIZZO PIU' ALTO BASIC  ( 55 / 56 ) 40960
```

I risultati che vedi si riferiscono alla situazione della memoria quando è presente questo programma. Esso non definisce variabili con indice (chiamate ARRAY) e quindi i relativi puntatori di inizio e fine sono allo stesso valore, puntano subito dopo la fine delle variabili.
Come vedi il più alto indirizzo del Basic è 40960 e il programma inizia in 2049. La zona dedicata al corpo delle stringhe comincia all'indirizzo 40959 e viene usata per indirizzi decrescenti. Le stringhe hanno le loro testate di definizione tra le altre variabili; nella prima zona se sono variabili stringa singole, nella seconda se sono variabili stringa con indice.

Segue un altro listato, relativo a un programma ottenuto modificando il precedente con l'aggiunta di un sottoprogramma in 200, che opera un dimensionamento e definisce due nuove stringhe; e con la modifica della linea 3, dove è stata aggiunta la chiamata al sottoprogramma. Al listato seguono i risultati su stampante.

```
1 REM PUNTBASMOD
2 INPUT"RISULTATI SU STAMPANTE? (S/N) ";R$
3 PRINT:GOSUB200:IFR$="N"THEN8
```

```
4 OPEN4,4
5 PRINT#4,"***CONTENUTO PUNTATORI A 2 BYTE***"
6 PRINT#4
8 D$="":READD$,B,A
9 IFD$="*"THEN40
10 Z=256*PEEK(A)+PEEK(B)
20 PRINTD$;SPC(2);"(";B;"/";A;")";Z
25 IFR$="N"THEN35
30 PRINT#4,D$;SPC(2);"(";B;"/";A;")";Z
31 PRINT#4
35 GOTO8
40 IFR$="S"THENCLOSE4
45 STOP
100 DATA"INIZIO PROGRAMMA BASIC",43,44
101 DATA"INIZIO VARIABILI BASIC",45,46
102 DATA"INIZIO ARRAY BASIC",47,48
103 DATA"FINE ARRAY BASIC + 1",49,50
104 DATA"IND. FONDO AREA STRINGHE",51,52
105 DATA"PUNTATORE STRINGHE",53,54
106 DATA"INDIRIZZO PIU' ALTO BASIC",55,56
107 DATA"*",0,0
200 DIMN(20):M$="UNA STRINGA"
201 N$="UN'ALTRA STRINGA"
202 RETURN
```

RISULTATI PROGRAMMA

```
***CONTENUTO PUNTATORI A 2 BYTE***

INIZIO PROGRAMMA BASIC  ( 43 / 44 ) 2049

INIZIO VARIABILI BASIC  ( 45 / 46 ) 2677

INIZIO ARRAY BASIC  ( 47 / 48 ) 2726

FINE ARRAY BASIC + 1  ( 49 / 50 ) 2838

IND. FONDO AREA STRINGHE  ( 51 / 52 ) 40959

PUNTATORE STRINGHE  ( 53 / 54 ) 40960

INDIRIZZO PIU' ALTO BASIC  ( 55 / 56 ) 40960
```

Come puoi vedere, avendo allungato il programma, l'inizio delle variabili si è spostato. Inoltre, avendo definito degli array i due puntatori relativi hanno valori diversi.

Se esegui in immediato il comando SYS 64738, cioè ripristini le condizioni iniziali del calcolatore, e, sempre in immediato vai a leggere con il comando PEEK il valore dei puntatori:

PRINT PEEK(k)+256*PEEK(k+1)

ponendo per k e k+1 i valori visti troverai che:

43/44	contengono	2049
45/46	"	2051
47/48	"	2051
49/50	"	2051
51/52	"	40960
53/54	"	0
55/56	"	40960

La figura 1.1 esemplifica l'uso della memoria da parte di un programma.

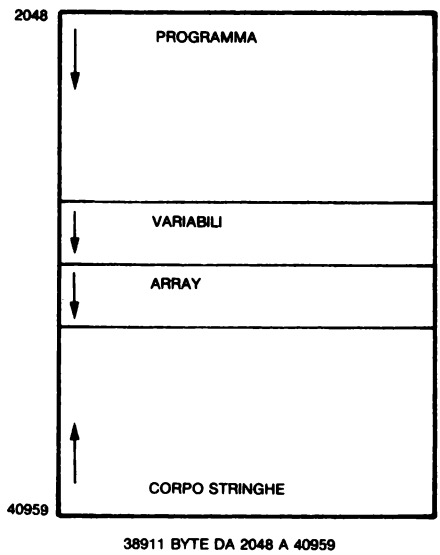


Figura 1.1 Utilizzo memoria RAM programma

Riportiamo un esempio abbastanza interessante di uso avanzato del linguaggio Basic.

I puntatori contenuti nei byte 63/64 e 65/66 controllano la gestione dei dati immagazzinati con le frasi DATA; il primo contiene il numero della linea DATA in uso e il secondo contiene l'indirizzo di memoria dell'elemento disponibile. Il sistema consente con l'istruzione RESTORE di riposizionarsi all'inizio dei dati. Se si desidera imparare a riposizionarsi in un punto desiderato della lista dei dati si può considerare il programma che segue, nel quale si leggono con delle istruzioni PEEK i contenuti dei puntatori in determinati momenti e più avanti si ripristinano i valori letti e memorizzati per riportarsi nella situazione precedente.

```
1 REM RESTORE
10 DATA0,1,2,3,4,5,6,7,8,9
11 DATA10,11,12,13,14,15,16,17,18,19
12 DATA20,21,22,23,24,25,26,27,28,29
15 OPEN4,4:CMD4
20 PRINT"VALORI INIZIALI"
25 GOSUB100
27 PRINT"LETTURA 10 NUMERI"
30 FORK=0TO9:READW:PRINTW;:NEXTK:PRINT
35 PRINT"DOPO 10 READ":GOSUB100:GOSUB150
37 PRINT"LETTURA ALTRI 5 NUMERI"
40 FORK=0TO4:READW:PRINTW;:NEXTK:PRINT
45 PRINT"DOPO ALTRI 5 READ":GOSUB100:GOSUB160
50 PRINT"DOPO RIPOSIZIONAMENTO  INIZIO LINEA 11"
53 GOSUB100:GOSUB150
54 PRINT"LETTURA 5 NUMERI"
55 FORK=0TO4:READW:PRINTW;:NEXTK:PRINT:GOSUB150
57 PRINT"LETTURA ULTIMI"
59 FORK=0TO7:READW:PRINTW;:NEXTK:PRINT
60 FORK=0TO6:READW:PRINTW;:NEXTK:PRINT
61 PRINT"VALORI FINALI PUNTATORI":GOSUB100
63 PRINT"RIPOSIZIONAMENTO META' LINEA 11"
64 GOSUB160:GOSUB100
65 PRINT"LETTURA 5 NUMERI"
70 FORK=0TO4:READW:PRINTW;:NEXTK:PRINT
99 PRINT#4:CLOSE4:STOP
100 X=PEEK(63)+256*PEEK(64)
101 Y=PEEK(65)+256*PEEK(66)
102 PRINT"PUNTATORE 63/64:  ";X
103 PRINT"PUNTATORE 65/66:  ";Y:RETURN
150 Z1%=PEEK(63):Z2%=PEEK(64)
151 Z3%=PEEK(65):Z4%=PEEK(66):RETURN
160 POKE63,Z1%:POKE64,Z2%
161 POKE65,Z3%:POKE66,Z4%:RETURN
```

RISULTATI PROGRAMMA RESTORE

```
VALORI INIZIALI
PUNTATORE 63/64:  0
PUNTATORE 65/66:  2048
LETTURA 10 NUMERI
 0  1  2  3  4  5  6  7  8  9
DOPO 10 READ
PUNTATORE 63/64:  10
PUNTATORE 65/66:  2087
LETTURA ALTRI 5 NUMERI
 10  11  12  13  14
DOPO ALTRI 5 READ
PUNTATORE 63/64:  11
PUNTATORE 65/66:  2107
DOPO RIPOSIZIONAMENTO  INIZIO LINEA 11
PUNTATORE 63/64:  10
PUNTATORE 65/66:  2087
LETTURA 5 NUMERI
 10  11  12  13  14
LETTURA ULTIMI
 15  16  17  18  19  20  21  22
 23  24  25  26  27  28  29
VALORI FINALI PUNTATORI
PUNTATORE 63/64:  12
PUNTATORE 65/66:  2157
RIPOSIZIONAMENTO META' LINEA 11
PUNTATORE 63/64:  11
PUNTATORE 65/66:  2107
LETTURA 5 NUMERI
 15  16  17  18  19
```

Il programma è stato caricato e lanciato subito dopo l'accensione del calcolatore; se lo fai girare più volte partendo da una situazione diversa potrai trovare una differenza nel contenuto iniziale del puntatore in 63/64; infatti esso non viene azzerato per effetto del comando RUN.
Il sottoprogramma in 150 memorizza i 4 byte in Z1%, Z2%, Z3% e Z4%; quello in 160 ripristina i valori dei puntatori prelevandoli dalle 4 variabili citate.
Il sottoprogramma in 100 calcola i valori dei puntatori e li stampa.

1.2 VARIABILI IN MEMORIA

L'interprete Basic assegna lo spazio di memoria alle variabili al momento della loro definizione.

La definizione delle variabili singole si ha quando esse sono citate la prima volta nel programma in una delle seguenti condizioni:
.. compaiono a sinistra di un uguale;
.. sono listate dopo la parola chiave INPUT;
.. sono listate dopo la parola chiave READ.

La definizione delle variabili con indice si ha:
.. quando si definiscono con la frase DIM;
.. quando viene citato la prima volta uno degli elementi e si ha un dimensionamento implicito a 10.
Il dimensionamento produce l'assegnazione del valore zero alle variabili numeriche, e del valore stringa-nulla alle variabili stringa.

Ricordiamo che il nome di una variabile può essere al massimo di due caratteri, di cui il primo deve essere una lettera e il secondo può essere una lettera o una cifra. Per ogni variabile singola il sistema usa 7 byte, i primi 2 servono per il nome che è considerato sempre di 2 caratteri, gli altri 5 sono usati in modo diverso a seconda del tipo della variabile. Nella descrizione delle variabili chiamiamo byte 1 il primo a sinistra e byte 7 l'ultimo a destra. Inoltre con HI intendiamo riferirci al valore del byte più significativo e con LO al valore del byte meno significativo.

VARIABILE INTERA: nome di al massimo 2 caratteri seguiti dal suffisso %.
.. byte 1: codice ASCII primo carattere nome + 128;
.. byte 2: codice ASCII secondo carattere + 128, se il nome è di un solo carattere, compare solo 128;
.. byte 3: byte HI del numero intero;
.. byte 4: byte LO del numero intero;
.. byte 5: non usato contiene 0;
.. byte 6: non usato contiene 0;
.. byte 7: non usato contiene 0.

VARIABILE REALE: nome di al massimo 2 caratteri senza suffisso.
.. byte 1: codice ASCII del primo caratere del nome;
.. byte 2: codice ASCII del secondo carattere del nome, o 0 se il nome è di un solo carattere;
.. byte 3: esponente del numero floating point;
.. byte 4: byte HI della mantissa del numero;

.. byte 5: byte successivo della mantissa;
.. byte 6: byte successivo della mantissa;
.. byte 7: byte LO della mantissa.

VARIABILE STRINGA: nome di al massimo 2 caratteri con il suffisso $.
.. byte 1: codice ASCII del primo carattere del nome;
.. byte 2: codice ASCII del secondo carattere del nome + 128, o solo 128 se il nome è di un solo carattere;
.. byte 3: contatore dei caratteri che compongono la stringa, massimo 255;
.. byte 4: byte LO del puntatore al corpo della stringa;
.. byte 5: byte HI del puntatore al corpo della stringa;
.. byte 6: non usato contiene 0;
.. byte 7: non usato contiene 0.
Nel momento in cui viene definita la prima volta la variabile stringa vengono assegnati questi 7 byte e il puntatore al corpo della stringa ha il valore che è possibile in quel momento, in dipendenza dalla situazione della zona di memoria assegnata ai corpi delle stringhe. Quando la stessa variabile stringa riceve un nuovo contenuto, in questi byte di testata vengono modificati: il byte 3, contenente il numero di caratteri e che quindi può variare, e i byte 4 e 5 che definiscono il puntatore a una nuova posizione di memoria. La memoria viene assegnata dinamicamente ai corpi delle stringhe durante l'uso del programma. Quando, dopo molti cambiamenti, la zona di memoria sembra esaurita, il sistema fa una operazione di compattamento, cioè sposta tutti i corpi delle stringhe verso il fondo della memoria, infatti si creano continuamente dei buchi, via via che si hanno le nuove assegnazioni. In tale modo il sistema si procura di nuovo memoria per i corpi delle stringhe; l'operazione viene chiamata "garbage collection". Può tuttavia succedere che durante l'uso di un programma si esaurisca la memoria disponibile per i corpi delle stringhe; tutte le stringhe vengono scritte alla massima lunghezza. L'unico rimedio in tale situazione è quello di porre dei limiti alle lunghezze delle stringhe, tagliandole quando vengono ricevute.
Durante l'esecuzione di un programma puoi avvertire delle soste inconsuete; esse dipendono dal fatto che il calcolatore ha esaurito la zona di memoria per i corpi delle stringhe e opera il compattamento.

Per le VARIABILI CON INDICE vale la stessa regola di quelle singole per la formazione dei nomi; non possono esserci confusioni dato che sono memorizzate in una zona a parte. Il numero dei byte occupati dipende dal numero delle dimensioni e dal valore di ogni dimensione. Si ha sempre una testata di 5 byte, seguita da una coppia di byte per ogni dimensione e da:
.. 2 byte per ogni elemento per le variabili intere;
.. 5 byte per ogni elemento per le variabili decimali (floating-point);
.. 3 byte per ogni elemento per le variabili stringa, più, in altra zona di memoria, un byte per ogni carattere di ogni elemento.

ARRAY DI VARIABILI PER NUMERI INTERI: nome di al massimo due caratteri seguito dal suffisso %.
.. byte 1: codice ASCII del primo carattere del nome + 128;
.. byte 2: codice ASCII del secondo carattere del nome + 128 o solo 128;
.. byte 3: byte LO del contatore del numero totale dei byte occupati;
.. byte 4: byte HI del contatore del numero totale dei byte occupati;
.. byte 5: numero delle dimensioni;
.. una coppia di byte come contatore (HI-LO) del numero degli elementi dell'ultima dimensione (valore massimo dell'indice +1);
.. una coppia di byte come contatore del numero degli elementi della penultima dimensione;
...
...
.. una coppia di byte come contatore del numero degli elementi della prima dimensione;
.. una coppia di byte (HI-LO) per ogni elemento.

Come vedi usando un array per definire le variabili intere si ha un risparmio di memoria rispetto alla definizione dello stesso numero di variabili come variabili singole. Supponiamo di definire 10 variabili singole intere e un array di 10 elementi interi. Nel primo caso occupiamo 7x10=70 byte; nel secondo 5+2x10+2x10=45 byte. Su un numero elevato di variabili questo può essere conveniente, anche se nel computo si deve tenere presente che, ogni volta che si richiama una variabile intera nel programma, si usano 2 o 3 byte per citare il nome della singola, mentre per quella con indice, altre ai 2 o 3 byte del nome ne servono 2 per aprire e chiudere le parentesi e da 1 a, diciamo, 4 per l'indice (pensiamo a un array con un solo indice).

ARRAY DI VARIABILI PER NUMERI FLOATING-POINT: nome di al massimo due caratteri senza suffisso.
Vale quanto detto per le variabili intere, salvo per il nome che è espresso come abbiamo visto per quelle floating-point singole, e per il numero di byte occupato per ogni elemento che sale a 5.
In questo caso non si ha risparmio di memoria rispetto alla definizione delle variabili singole; si deve ricorrere a questo tipo di array quando è utile per lo svolgimento della programmazione.

ARRAY DI VARIABILI PER STRINGHE: nome di al massimo due caratteri seguito dal suffisso $.
Vale quanto detto per le variabili intere, salvo per il nome che è espresso come abbiamo visto per le variabili stringa singole, e per il numero di byte occupato per ogni elemento che sale a 3. Questi 3 byte sono così utilizzati: il primo come contatore del numero di caratteri dell'elemento, che può arrivare al massimo a 255; gli altri due byte (LO-HI) del puntatore al CORPO della stringa che si trova nella zona di memoria dedicata allo scopo.

Per vedere quanto abbiamo detto in memoria, abbiamo preparato un programma che crea variabili di ogni tipo e, servendosi dei puntatori, va a leggere e stampare i contenuti delle diverse zone della memoria.

```
1 REM VARINMEM
3 M=0:N=0:Y=0:Z=0
5 DIMD(6),E%(5,4),F$(5,3,2)
7 A=0:FORK=1TO50 A=A+1:NEXTK
9 FORK=0TO6:D(K)=K*3:NEXTK
11 B%=0:FORI=1TO50STEP2:B%=B%+2:NEXTI
13 FORK=0TO5:FORI=0TO4
15 E%(K,I)=K*1+9:NEXTI,K
17 C$="FINE"
19 FORK=0TO5:FORJ=0TO3:FORI=0TO2
21 READY$:F$(K,J,I)=C$+Y$
23 NEXTI,J,K
25 Y=PEEK(45)+256*PEEK(46)
27 OPEN4,4:CMD4
29 PRINT:PRINT"VARIABILI":M=Y
31 PRINTM;"**"
33 FORI=0TO10:FORN=0TO6
35 Y=PEEK(45)+256*PEEK(46)
37 PRINTPEEK(M+N);"  ";:NEXTN:PRINT:PRINT
39 M=M+7:NEXTI
41 Y=PEEK(47)+256*PEEK(48)
43 Z=PEEK(49)+256*PEEK(50)
45 PRINT:PRINT"ARRAY":PRINTY;"**":K=Y
51 M=PEEK(K+2)+256*PEEK(K+3)
57 I=0:FORJ=KTOK+M-1
59 PRINTPEEK(J);" ";:I=I+1
61 IFI=7THENPRINT:I=0
63 NEXTJ:K=K+M:IFK=ZTHEN67
65 PRINT:GOTO51
67 PRINT:PRINT"FINE ARRAY: ";PEEK(Z)
79 PRINT:PRINT:PRINT"CORPO STRINGHE":PRINT
81 Y=PEEK(51)+256*PEEK(52)
82 Z=PEEK(55)+256*PEEK(56)
83 I=0:FORK=YTOY+50:PRINTK;"*";PEEK(K);
84 I=I+1:IFI=3THENPRINT:I=0
85 NEXTK
86 I=0:FORK=Z-50TOZ:PRINTK;"*";PEEK(K);
87 I=I+1:IFI=3THENPRINT:I=0
88 NEXTK
```

```
100 PRINT#4:CLOSE4:STOP
110 DATAAAA,BBB,CCC,DDD,EEE,FFF,GGG,III,LLL
111 DATAA,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T
112 DATAU,V,W,X,Y,Z,A1,B1,C1,D1,E1,F1,G1,H1,I1
113 DATAJ1,K1,L1,M1,N1,O1,P1,Q1,R1,S1,T1,U1,V1
114 DATAW1,X1,Y1,Z1,ABC,DEF,GHI,JKL,MNO,PQR,STU
115 DATAVWX,YZZ,PIPPO,PLUTO
```

RISULTATI PROGRAMMA VARINMEM

```
VARIABILI
 3109 **
 77    0    140    66    80    0    0

 78    0    130    64    0    0    0

 89    0    140    66    80    0    0

 90    0    0    0    0    0    0

 65    0    134    72    0    0    0

 75    0    131    64    0    0    0

 194    128    0    50    0    0    0

 73    0    131    96    0    0    0

 67    128    4    191    8    0    0

 74    0    131    0    0    0    0

 89    128    5    29    12    0    0


ARRAY
 3186 **
 68    0    42    0    1    0    7
 0    64    0    0    0    130    64
 0    0    0    131    64    0    0
 0    132    16    0    0    0    132
 64    0    0    0    132    112    0
 0    0    133    16    0    0    0
```

```
197   128   69   0   2   0   5
0   6   0   9   0   10   0
11   0   12   0   13   0   14
0   9   0   10   0   11   0
12   0   13   0   14   0   9
0   10   0   11   0   12   0
13   0   14   0   9   0   10
0   11   0   12   0   13   0
14   0   9   0   10   0   11
0   12   0   13   0   14
70   128   227   0   3   0   3
0   4   0   6   7   249   159
5   173   159   5   113   159   6
51   159   6   235   158   6   163
158   7   228   159   5   158   159
5   98   159   6   33   159   6
217   158   7   142   158   7   207
159   5   143   159   5   83   159
6   15   159   6   199   158   7
121   158   5   188   159   5   128
159   5   68   159   6   253   158
6   181   158   7   100   158   7
242   159   5   168   159   5   108
159   6   45   159   6   229   158
7   156   158   7   221   159   5
153   159   5   93   159   6   27
159   6   211   158   7   135   158
7   200   159   5   138   159   5
78   159   6   9   159   6   193
158   7   114   158   5   183   159
5   123   159   5   63   159   6
247   158   6   175   158   9   91
158   7   235   159   5   163   159
5   103   159   6   39   159   6
223   158   7   149   158   7   214
159   5   148   159   5   88   159
6   21   159   6   205   158   7
128   158   7   193   159   5   133
159   5   73   159   6   3   159
6   187   158   7   107   158   5
178   159   5   118   159   6   57
159   6   241   158   6   169   158
9   82   158
FINE ARRAY:  0
```

CORPO STRINGHE

```
40530 * 70  40531 * 73  40532 * 78
40533 * 69  40534 * 80  40535 * 76
40536 * 85  40537 * 84  40538 * 79
40539 * 70  40540 * 73  40541 * 78
40542 * 69  40543 * 80  40544 * 73
40545 * 80  40546 * 80  40547 * 79
40548 * 70  40549 * 73  40550 * 78
40551 * 69  40552 * 89  40553 * 90
40554 * 90  40555 * 70  40556 * 73
40557 * 78  40558 * 69  40559 * 86
40560 * 87  40561 * 88  40562 * 70
40563 * 73  40564 * 78  40565 * 69
40566 * 83  40567 * 84  40568 * 85
40569 * 70  40570 * 73  40571 * 78
40572 * 69  40573 * 80  40574 * 81
40575 * 82  40576 * 70  40577 * 73
40578 * 78  40579 * 69  40580 * 77
40910 * 73  40911 * 70  40912 * 73
40913 * 78  40914 * 69  40915 * 71
40916 * 71  40917 * 71  40918 * 70
40919 * 73  40920 * 78  40921 * 69
40922 * 70  40923 * 70  40924 * 70
40925 * 70  40926 * 73  40927 * 78
40928 * 69  40929 * 69  40930 * 69
40931 * 69  40932 * 70  40933 * 73
40934 * 78  40935 * 69  40936 * 68
40937 * 68  40938 * 68  40939 * 70
40940 * 73  40941 * 78  40942 * 69
40943 * 67  40944 * 67  40945 * 67
40946 * 70  40947 * 73  40948 * 78
40949 * 69  40950 * 66  40951 * 66
40952 * 66  40953 * 70  40954 * 73
40955 * 78  40956 * 69  40957 * 65
40958 * 65  40959 * 65  40960 * 148
```

Il programma definisce 11 variabili singole; esse vengono assegnate nel seguente ordine: M, N, Y, Z, A, K, B%, I, C$, J, Y$. Inoltre esso definisce 3 array, nel seguente ordine: D, E%, F$.
Per poter lavorare abbiamo dovuto definire all'inizio del programma, prima di andare a leggere il valore dei puntatori tutte le variabili, infatti (se osservi la Figura

1.1, te ne rendi subito conto), se si aggiunge una variabile singola, tutti gli array devono essere spostati in giù; questa è anche una perdita di tempo. Osserva la linea 3; in essa vengono definite 4 variabili singole, assegnando ad esse il valore zero. E' consigliabile fare all'inizio del programma sempre tutte le assegnazioni delle variabili singole, al limite ponendole a zero, poi passare a definire gli array. Operando in tale modo le variabili restano posizionate in memoria allo stesso posto per tutta la durata del programma; variano in modo dinamico solo le posizioni dei corpi delle stringhe.
Nel programma dalla linea 3 alla 23 si ha la creazione di tutte le variabili, anche di quelle che vengono usate come contatori per controllare i cicli FOR.
Per assegnare valori alla matrice di stringhe che ha 72 elementi (6x4x3=72), ci siamo serviti di frasi DATA dalla linea 110 alla 115, e del ciclo con READ dalla linea 19 alla linea 23. Nota la chiusura dei tre cicli FOR concatenati con una sola parola chiave NEXT seguita dai nomi delle 3 variabili di controllo.
Dalla linea 25 alla linea 39 si stampano i contenuti dei byte usati per le variabili singole. Si ricava dai byte 45 e 46 il valore del puntatore (3109)e si stampa seguito da due asterischi. Poi vengono stampati su ogni riga i 7 byte di ognuna delle 11 variabili singole. Puoi controllare quanto abbiamo detto all'inizio di questo paragrafo. Osserva le due linee che si riferiscono a stringhe, la terzultima e l'ultima, nella parte dei risultati intitolata VARIABILI. Vedi per C$ il nome: 67, 128, poi il 4 del terzo byte dice che la stringa ha 4 caratteri (FINE), i due byte successivi sono rispettivamente i byte LO e HI del puntatore al corpo della stringa (8x256+191=2239) che si trova incorporata nel programma alla linea 17. Per Y$ invece, che risulta essere di 5 caratteri, infatti l'ultima stringa trattata con READ è "PLUTO", si trova come valore del puntatore 12x256+29=3101, questo è l'indirizzo di memoria interno al programma dove inizia la parola "PLUTO" alla fine della linea 115. Dopo si ha lo zero di fine istruzione al byte 3106, i due byte 3107 e 3108 contengono i due zeri di fine programma e in 3109 iniziano le variabili.
Dalla linea 41 alla linea 67 si stampano gli array. Si calcola l'inizio degli array per mezzo del puntatore nei byte 47 e 48 e la fine più uno per mezzo del puntatore nei byte 49 e 50. Alla linea 51 si calcola M usando il terzo e il quarto byte; esso è il numero dei caratteri occupati dall'array in questa zona. Per il primo array D(6) sono occupati in tutto 42 byte. Il quinto byte dice che si ha una dimensione, il sesto e il settimo byte (prima HI e poi LO) dicono che l'array ha 7 elementi. Seguono poi 5 byte per ogni elemento; se non riesci a ritrovare i numeri che devono essere: 0, 3, 6, 9, 12, 15, 18, vai a rivedere nel Capitolo 7 la parte dove vengono svolti i calcoli relativi alla mantissa dei numeri floating-point.
Per E%(5,4) trovi dopo il nome, che occupa in tutto 69 byte, che ha 2 dimensioni, che l'ultima dimensione ha 5 elementi, che la prima dimensione ha 6 elementi; dopo trovi gli elementi che occupano 2 byte ciascuno.
Per F$(5,3,2), che occupa in tutto 227 byte, fissiamo la nostra attenzione su qualche elemento. Il primo (byte 12,13 e 14) ha 7 caratteri e il corpo della stringa si trova in 159x256+249=40953; lo puoi vedere in fondo ai risultati. Partendo dal byte 40953,

troviamo i codici: 70, 73, 78, 69, 65, 65, 65, che corrispondono a "FINEAAA". Ti rendi conto del modo come vengono sistemate le stringhe in memoria; questa è stata assegnata per prima e sta sul fondo della memoria. Consideriamo l'ultimo elemento (9, 82, 158), esso ha 9 caratteri e la stringa si trova in 158x256+82=40530; puoi vederla all'inizio della parte dei risultati intitolata CORPO STRINGHE. Partendo dal byte 40530 troviamo: 70, 73, 78, 69, 80, 76, 85, 84, 79, che corrisponde a "FINEPLUTO", ultima stringa assegnata.
Nelle linee da 79 a 88 si stampa una parte del corpo delle stringhe e precisamente gli ultimi 51 byte e i primi 51 byte.
Osserva che tra le variabili singole si trovano anche le variabili di controllo dei cicli FOR; esse si presentano come variabili reali e non si differenziano dalle altre variabili dello stesso tipo. Quando viene eseguito un ciclo FOR le informazioni necessarie per gestirlo vengono memorizzate nell'area STACK del sistema, mentre le variabili di controllo restano in zona variabili.

Nella definizione delle variabili singole si ha spreco di memoria per gli interi e per le stringhe. Infatti per gli interi restano inutilizzati gli ultimi 3 byte dei 7 occupati e per le stringhe gli ultimi 2. Abbiamo preparato il programma UTILIZZO per farti vedere come puoi andare a scrivere qualcosa nei byte inutilizzati senza disturbare le variabili. Naturalmente devi procurarti gli indirizzi dei byte e scriverci dentro usando l'istruzione POKE.

```
1 REM UTILIZZO
10 REM DEFINIZIONE 2 VAR. INTERE
20 A%=5:B%=7
25 REM DEFINIZIONE 2 VAR. STRINGA
30 A$="PROVA":B$="RISPARMIO"
35 X=PEEK(45)+256*PEEK(46)
40 Y%=A%:GOSUB100:Y=X+4:GOSUB101
45 Y%=B%:GOSUB100:Y=X+11:GOSUB101
50 Y%=ASC(A$):POKEX+19,Y%
55 Y%=ASC(B$):POKEX+26,Y%
56 OPEN4,4:CMD4
60 PRINT"A%=";A%,"B%=";B%
65 PRINT"A$="A$,"B$="B$
67 PRINT"VARIABILI IN MEMORIA"
69 FORZ=0TO21STEP7
70 FORY=0TO6:PRINTPEEK(Z+Y+X);
71 NEXTY:PRINT:NEXTZ
75 PRINT#4:CLOSE4:STOP
100 Y%=Y%↑2:RETURN
101 POKEY,INT(Y%/256)
102 POKE(Y+1),Y%-PEEK(Y):RETURN
```

RISULTATI PROGRAMMA UTILIZZO

```
A%= 5              B%= 7
A$=PROVA             B$=RISPARMIO
VARIABILI IN MEMORIA
 193  128  0  5  0  25  0
 194  128  0  7  0  49  0
 65  128  5  103  8  80  0
 66  128  9  114  8  82  0
```

Il programma definisce per prime le 4 variabili che interessano; esse sono A%, B%, A$ e B$. In tale modo le variabili occupano quattro gruppi di 7 byte subito all'inizio dell'area variabili. Il sottoprogramma in 100 calcola il quadrato del numero che sta in Y% e lo pone ancora in Y%. Il sottoprogramma in 101 va a scrivere Y% nei 2 byte di indirizzo Y e Y+1.
Le variabili vengono stampate nel modo normale anche dopo aver usato i byte di coda e, come vedi, non sono disturbate. Dopo vengono mostrati i contenuti della memoria e puoi vedere i quadrati dei due numeri: 25 e 49. Per le due stringhe invece vedi il codice ASCII del primo carattere scritto nel penultimo byte.
Questo è un uso abbastanza sofisticato della memoria del calcolatore e abbiamo preparato l'esempio per far comprendere come lavora il sistema.

1.3 PROGRAMMA IN MEMORIA

Il programma Basic inizia al byte 2049, puntato dai due byte 43 e 44 della pagina zero.
La struttura delle istruzioni è la seguente:

.. 2 byte di LINK (legamento), che contengono l'indirizzo della linea di programma successiva; l'ultima istruzione del programma ha i due byte di link a zero binario (tutti bit zero); il primo byte è il LO, il secondo lo HI;
.. 2 byte contenenti il numero della linea Basic; il primo LO e il secondo HI;
.. seguono i byte che contengono la linea Basic, nella quale le parole chiave sono compresse e vengono espresse dal codice in un byte, mentre tutte le altre parti componenti la linea sono espresse carattere per carattere;
.. la linea termina con un byte a zero binario.

Da quanto detto risulta che tre zeri binari vicini segnalano la fine del programma.
Se la linea Basic contiene degli spazi, essi vengono mantenuti con spreco di memoria. Non viene conservato lo spazio tra il numero di linea e il primo carattere della frase; questo spazio viene aggiunto in fase di LIST del programma.

Segue un breve programma che stampa se stesso, a scopo dimostrativo.

```
1 REM PRGINMEM
5 OPEN4,4:CMD4:PRINT"PROGRAMMA IN MEMORIA":PRINT
10 X=256*PEEK(44)+PEEK(43)
15 Y1=PEEK(X):Y2=PEEK(X+1)
17 IFY1+Y2=0THENPRINTX;"**";Y1;Y2:GOTO60
20 PRINTX;"**";Y1;Y2;"LINK=";Y2*256+Y1
25 Y1=PEEK(X+2):Y2=PEEK(X+3)
30 PRINT"NUMERO LINEA: ";Y2*256+Y1
35 X=X+4:C=0
37 Y1=PEEK(X):IFY1=0THEN55
40 PRINTY1;"   ";:X=X+1:C=C+1
45 IFC=6THENPRINT:C=0
50 GOTO37
55 PRINTY1:X=X+1:GOTO15
60 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA PRGINMEM

```
PROGRAMMA IN MEMORIA

 2049 ** 16  8 LINK= 2064
NUMERO LINEA:  1
 143     32     80     82     71     73
 78     77     69     77     0
 2064 ** 54  8 LINK= 2102
NUMERO LINEA:  5
 159     52     44     52     58     157
 52     58     153     34     80     82
 79     71     82     65     77     77
 65     32     73     78     32     77
 69     77     79     82     73     65
 34     58     153     0
 2102 ** 76  8 LINK= 2124
NUMERO LINEA:  10
 88     178     50     53     54     172
 194     40     52     52     41     170
 194     40     52     51     41     0
 2124 ** 98  8 LINK= 2146
```

```
NUMERO LINEA:  15
 89      49      178      194      40      88
 41      58      89      50      178      194
 40      88      170      49      41      0
 2146 ** 129  8 LINK= 2177
NUMERO LINEA:  17
 139      89      49      170      89      50
 178      48      167      153      88      59
 34      42      42      34      59      89
 49      59      89      50      58      137
 54      48      0
 2177 ** 165  8 LINK= 2213
NUMERO LINEA:  20
 153      88      59      34      42      42
 34      59      89      49      59      89
 50      59      34      76      73      78
 75      61      34      59      89      50
 172      50      53      54      170      89
 49      0
 2213 ** 189  8 LINK= 2237
NUMERO LINEA:  25
 89      49      178      194      40      88
 170      50      41      58      89      50
 178      194      40      88      170      51
 41      0
 2237 ** 221  8 LINK= 2269
NUMERO LINEA:  30
 153      34      78      85      77      69
 82      79      32      76      73      78
 69      65      58      32      34      59
 89      50      172      50      53      54
 170      89      49      0
 2269 ** 235  8 LINK= 2283
NUMERO LINEA:  35
 88      178      88      170      52      58
 67      178      48      0
 2283 ** 0  9 LINK= 2304
NUMERO LINEA:  37
 89      49      178      194      40      88
 41      58      139      89      49      178
 48      167      53      53      0
 2304 ** 27  9 LINK= 2331
```

```
NUMERO LINEA:  40
 153     89     49     59     34     32
 32     32     34     59     58     88
 178     88     170     49     58     67
 178     67     170     49     0
 2331 ** 42  9 LINK= 2346
NUMERO LINEA:  45
 139     67     178     54     167     153
 58     67     178     48     0
 2346 ** 50  9 LINK= 2354
NUMERO LINEA:  50
 137     51     55     0
 2354 ** 68  9 LINK= 2372
NUMERO LINEA:  55
 153     89     49     58     88     178
 88     170     49     58     137     49
 53     0
 2372 ** 80  9 LINK= 2384
NUMERO LINEA:  60
 152     52     58     160     52     58
 144     0
 2384 ** 0  0
```

Come puoi vedere abbiamo stampato le linee di programma andando a capo ad ogni linea e indicando l'indirizzo di inizio di ogni istruzione, che coincide con il numero contenuto nei due byte di link. Puoi andare a cercare nelle tabelle dei codici delle parole chiave e dei caratteri e ricostruire le linee del programma.

Il sistema usa dei puntatori per seguire lo svolgimento del programma, e precisamente:

byte 57/58 numero linea corrente
byte 59/60 numero ultima linea del programma precedente
byte 61/62 indirizzo linea corrente

più altri che puoi ritrovare nell'elenco delle variabili nel prossimo paragrafo.

Segue il programma NUMLINEE, che memorizza in una matrice di 5 righe e 3 colonne (non usiamo gli indici 0), lo stato dei tre puntatori citati, e dopo stampa la matrice.

```
1 REM NUMLINEE
5 DEFFNA(B)=PEEK(B)+256*PEEK(B+1)
6 Y=57:Z=59:T=61
7 OPEN4,4:CMD4:PRINT"CONTENUTO PUNTATORI"
8 PRINT
9 PRINT"57/58      59/60      61/62":PRINT
10 DIMX(5,3)
15 X(1,1)=FNA(Y)
16 X(1,2)=FNA(Z)
17 X(1,3)=FNA(T)
18 X(2,1)=FNA(Y)
19 X(2,2)=FNA(Z)
20 X(2,3)=FNA(T)
21 X(3,1)=FNA(Y)
22 X(3,2)=FNA(Z)
23 X(3,3)=FNA(T)
24 X(4,1)=FNA(Y)
25 X(4,2)=FNA(Z)
26 X(4,3)=FNA(T)
27 X(5,1)=FNA(Y)
28 X(5,2)=FNA(Z)
29 X(5,3)=FNA(T)
100 FORK=1TO5:FORJ=1TO3
105 PRINTX(K,J);"       ";:NEXTJ:PRINT
110 NEXTK
199 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA PRECEDENTE

```
CONTENUTO PUNTATORI

57/58      59/60      61/62

 15         0          2231
 18         0          2282
 21         0          2333
 24         0          2384
 27         0          2435
```

CONTENUTO PUNTATORI

57/58	59/60	61/62
15	199	2231
18	199	2282
21	199	2333
24	199	2384
27	199	2435

CONTENUTO PUNTATORI

57/58	59/60	61/62
15	10	2231
18	10	2282
21	10	2333
24	10	2384
27	10	2435

Abbiamo riportato 3 risultati per controllare il contenuto dei byte 59 e 60. Nel primo risultato il puntatore è zero; il programma è stato fatto girare subito dopo l'accensione del calcolatore. Nel secondo risultato il puntatore contiene 199, che è il numero di linea eseguito per ultimo facendo girare il programma la prima volta. Nel terzo risultato il puntatore contiene 10; esso è stato ottenuto così: dopo il comando RUN abbiamo premuto subito RUN/STOP e il programma si è arrestato con il messaggio BREAK IN LINE 10; subito dopo abbiamo dato ancora RUN.

Questa prova può anche non essere particolarmente interessante, ma serve a indicare un metodo per capire come si comporta il calcolatore.

Facciamo un breve cenno alla sistemazione in memoria dei programmi in linguaggio macchina. Essi occupano una serie di byte, il cui contenuto. binario è il programma in linguaggio macchina. Se non disponi di un Programma Assemblatore o di un Monitor o di un Caricatore Esadecimale, devi scrivere con POKE a partire dall'indirizzo voluto il contenuto dei byte, usando il corrispondente valore decimale.

1.4 MEMORIA DI LAVORO DEL SISTEMA

Le prime quattro pagine di memoria RAM sono usate dall'Interprete Basic e dal Sistema Operativo per lavorare. Abbiamo già visto l'utilizzo di alcuni puntatori servendoci di brevi programmi esemplificativi. Sarebbe molto interessante condurre uno studio completo sull'argomento, ma questo manuale diventerebbe troppo lungo. Qui riportiamo l'elenco dei byte delle pagine 0, 1, 2 e 3 della RAM con brevi spiegazioni sul loro utilizzo per tua comodità di consultazione. In alcuni casi potrai preparare dei programmi esempio, sulla scia di quelli suggeriti da noi, e arrivare a una comprensione più completa di specifici argomenti. Noi stessi in questo manuale e nei prossimi approfondiremo alcuni degli argomenti ai quali qui si fa solo un breve cenno.

Nella tabella che segue riportiamo su una riga il nome simbolico della variabile usata dal sistema (LABEL), l'indirizzo decimale del o dei byte, l'indirizzo esadecimale del o dei byte (senza H finale) e sulla riga seguente, ed eventualmente sulle successive, brevi spiegazioni. Quando i byte interessati sono più di uno riportiamo l'indirizzo iniziale e quello finale separati da una barra (/). In alcuni casi nelle spiegazioni compare un numero e poi tra parentesi un altro numero; il primo è decimale, il secondo la conversione in esadecimale.

PAGINE 0, 1, 2, 3 DELLA MEMORIA RAM

D6510 0 0000
Registro direzione dati del circuito 6510. Al momento dell'accensione del calcolatore contiene 47. Tale valore dà la configurazione di memoria riportata nella Figura 1.6 del Capitolo 1.

R6510 1 0001
Registro di Input/Output del circuito 6510. Al momento dell'accensione del calcolatore contiene 55.

2 0002
Non usato

ADRAY1 3/4 0003/0004
Vettore di salto per la conversione floating-point/intero.

ADRAY2 5/6 0005/0006
Vettore di salto per la conversione intero/floating-point.

CHARAC 7 0007
Byte di lavoro: carattere da ricercare.

ENDCHR	8	0008

Indicatore ricerca virgolette di chiusura stringhe.

TRMPOS	9	0009

Posizione carattere sullo schermo dopo TAB.

VERCK	10	000A

Indicatore per lettura programmi: 0=LOAD, 1=VERIFY.

COUNT	11	000B

Puntatore buffer input/numero dimensioni variabile con indici.

DIMFLG	12	000C

Indicatore dimensioni variabile con indice non definita da DIM.

VALTYP	13	000D

Tipo di dato: 255 (FF) per stringa, 0 (00) per numerico.

INTFLG	14	000E

Tipo di dato: 128 (80) per intero, 0 (00) per floating-point.

GARBFL	15	000F

Memoria lavoro uso multiplo.

SUBFLG	16	0010

Indicatore uso multiplo.

INPFLG	17	0011

Indicatore per operazione lettura: 0 (00) per INPUT, 64 (40) per GET, 152 (98) per READ.

TANSGN	18	0012

Segno per funzione TAN/confronto risultati.

	19	0013

Indicatore richiesta Input.

LINNUM	20/21	0014/0015

Memoria temporanea per numero intero.

TEMPPT	22	0016

Puntatore stack per stringa in elaborazione.

LASTPT 23/24 0017/0018
Indirizzo ultima stringa in elaborazione.

TEMPST 25/33 0019/0021
Stack per stringa in elaborazione.

INDEX 34/37 0022/0025
Area puntatori per programmi di utilità.

RESHO 38/42 0026/002A
Risultato moltiplicazione floating-point.

TXTTAB 43/44 002B/002C
Puntatore inizio programma Basic; di norma contiene 2049.

VARTAB 45/46 002D/002E
Puntatore inizio variabili programma Basic.

ARYTAB 47/48 002F/0030
Puntatore inizio Array programma Basic.

STREND 49/50 0031/0032
Puntatore fine Array + 1 (ind. byte successivo).

FRETOP 51/52 0033/0034
Puntatore all'indirizzo più alto della memoria utilizzata per i corpi delle stringhe.

FRESPC 53/54 0035/0036
Puntatore ultima stringa memorizzata.

MEMSIZ 55/56 0037/0038
Puntatore all'indirizzo più alto disponibile per il programma Basic; al momento dell'accensione contiene 40960.

CURLIN 57/58 0039/003A
Numero linea corrente del programma presente in memoria.

OLDLIN 59/60 003B/003C
Numero linea eseguita per ultima dal programma precedente, o numero della linea dell'ultimo BREAK del programma corrente.

OLDTXT 61/62 003D/003E
Indirizzo linea corrente del programma corrente (ind. byte dove inizia la linea che ha il numero di linea in CURLIN).

DATLIN 63/64 003F/0040
Numero linea del DATA in utilizzo.

DATPTR 65/66 0041/0042
Indirizzo del byte dove inizia una nuova linea DATA o indirizzo dell'elemento disponibile in una linea già in utilizzo.

INPPTR 67/68 0043/0044
Vettore per la routine di INPUT.

VARNAM 69/70 0045/0046
Nome variabile in uso nel programma Basic.

VARPNT 71/72 0047/0048
Puntatore alla variabile in uso nel programma Basic.

FORPNT 73/74 0049/004A
Puntatore alla variabile di controllo del ciclo FOR/NEXT.

75/96 004B/0060
Area di lavoro ad uso multiplo.

FACEXP 97 0061
Esponente (caratteristica) ACC 1 per calcolo numeri floating-point.

FACHO 98/101 0062/0065
Mantissa ACC 1 per calcolo numeri floating-point.

FACSGN 102 0066
Segno mantissa ACC 1.

SGNFLG 103 0067
Indicatore per la valutazione del segno.

BITS 104 0068
Indicatore Overflow per ACC 1.

ARGEXP 105 0069
Esponente ACC 2.

ARGHO 106/109 006A/006D
Mantissa ACC 2.

ARGSGN 110 006E
Segno ACC 2.

ARISGN 111 006F
Segno risultato confronto tra ACC 1 e ACC 2.

FACOV 112 0070
Ultimo byte (byte basso) ACC 1 usato per l'arrotondamento.

FBUFPT 113/114 0071/0072
Puntatore al buffer della cassetta.

CHRGET 115/138 0073/008A
Sottoprogramma che preleva il prossimo byte dal testo del programma Basic. Il byte 0079 (121) è il punto di entrata per prelevare nuovamente lo stesso byte del testo; esso ha il nome simbolico CHRGOT. Il nome simbolico TXTPTR che corrisponde ai due byte 122 e 123 (007A/007B) si riferisce al puntatore al byte corrente della linea del testo.

RNDX 139/143 008B/008F
Seme usato dalla funzione RND, espresso in floating-point.

STATUS 144 0090
Parola di stato usata dalle Routine KERNAL per Input/Output; nel programma Basic viene chiamata ST.

STKEY 145 0091
Indicatore per tasto STOP/tasto RVS.

SVXT 146 0092
Costante di controllo per velocità nastro.

VERCK 147 0093
Indicatore: 0=LOAD, 1=VERIFY.

C3PO 148 0094
Indicatore per Bus Seriale: carattere bufferizzato per Output.

BSOUR 149 0095
Carattere bufferizzato per Bus Seriale.

SYNO 150 0096
Numero per sincronizzazione cassetta.

151 0097
Area di lavoro.

LDTND 152 0098
Numero file aperti/Puntatore alla tabella dei file.

DFLTN 153 0099
Dispositivo di Input principale (default) 0, cioè tastiera.

DFLTO 154 009A
Dispositivo di Output principale (default) 3, cioè video.

RTY 155 009B
Carattere di parità del nastro.

DPSW 156 009C
Indicatore ricezione byte da nastro.

MSGFLG 157 009D
Indicatore modo operativo: 128 (80) per modo diretto, 0 (00) modo differito.

PTR1 158 009E
Indicatore errore passo 1 nastro.

PTR2 159 009F
Indicatore errore passo 2 nastro.

TIME 160/162 00A0/00A2
Orologio aggiornato automaticamente ogni sessantesimo di secondo.

163/164 00A3/00A4
Area di lavoro.

CNTDN 165 00A5
Contatore in decremento per sincronizzazione cassetta.

BUFPNT 166 00A6
Puntatore al buffer del nastro.

INBIT 167 00A7
Input bit da RS232/Cassetta.

BITCI 168 00A8
Contatore bit Input da RS232/Cassetta.

RINONE 169 00A9
Indicatore per RS232: controllo bit di START.

RIDATA 170 00AA
Buffer byte Input per RS232/Cassetta.

RIPRTY 171 00AB
Parità per Input RS232/Contatore corto cassetta.

SAL 172/173 00AC/00AD
Puntatore Buffer Nastro/ Scrolling video.

EAL 174/175 00AE/00AF
Indirizzo fine nastro/fine programma.

CMPO 176/177 00B0/00B1
Costanti per misura operazioni nastro.

TAPE1 178/179 00B2/00B3
Puntatore inizio buffer nastro.

BITTS 180 00B4
Contatore bit Output RS232/Cassetta.

NXTBIT 181 00B5
Prossimo bit da inviare per RS232/Indicatore fine nastro.

RODATA 182 00B6
Buffer Output RS232.

FNLEN 183 00B7
Lunghezza del nome del file corrente.

LA 184 00B8
Numero logico del file corrente.

SA 185 00B9
Indirizzo secondario file corrente.

FA 186 00BA
Numero logico periferica corrente.

FNADR	187/188	00BB/00BC

Puntatore al nome del file corrente.

ROPRTY	189	00BD

Parità Output per RS232/Cassetta.

FSBLK	190	00BE

Contatore blocco Input/Output cassetta.

MYCH	191	00BF

Buffer seriale.

CAS1	192	00C0

Arresto motore nastro.

STAL	193/194	00C1/00C2

Indirizzo partenza Input/Output.

MEMUSS	195/196	00C3/00C4

Memoria temporanea LOAD nastro.

LSTX	197	00C5

Tasto corrente premuto: CHR$(n); 0 per nessun tasto.

NDX	198	00C6

Numero caratteri presenti nel buffer della tastiera.

RVS	199	00C7

Indicatore stampa caratteri inversi: 1 per si, 0 per no.

INDX	200	00C8

Puntatore alla fine della linea logica in INPUT.

LXSP	201/202	00C9/00CA

Posizione X-Y del cursore all'inizio dell'Input.

SFDX	203	00CB

Indicatore tasto SHIFT premuto.

BLNSW	204	00CC

Abilitazione lampeggio cursore: 1 per lampeggio.

BLNCT	205	00CD

Contatore in decremento per lampeggio.

GDBLN 206 00CE
Carattere sotto il cursore.

BLNON 207 00CF
Indicatore ultima impostazione cursore: lampeggio si/no.

CRSW 208 00D0
Indicatore INPUT/GET da tastiera.

PNT 209/210 00D1/00D2
Puntatore linea corrente del video.

PNTR 211 00D3
Posizione cursore nella linea corrente.

QTSW 212 00D4
Indicatore: Editor in modo "tra virgolette"; 0 (00) per no.

LNMX 213 00D5
Lunghezza fisica linea video.

TBLX 214 00D6
Numero linea attuale del cursore.

215 00D7
Area di lavoro.

INSRT 216 00D8
Indicatore modo inserimento: se > 0 dà il numero inserimenti.

LDTB1 217/242 00D9/00F2
Tabella dei collegamenti per le linee video/Memoria di lavoro per Editor.

USER 243/244 00F3/00F4
Puntatore locazione corrente della RAM colore del video.

KEYTAB 245/246 00F5/00F6
Vettore tabella decodifica tastiera.

RIBUF 247/248 00F7/00F8
Puntatore Buffer Input RS232.

ROBUF 249/250 00F9/00FA
Puntatore Buffer Output RS232.

FREKZP 251/254 00FB/00FE
Spazio libero in pagina 0 per programmi utente.

BASKZP 255 00FF
Area lavoro Basic.

256/511 0100/01FF
STACK AREA del sistema (256 byte). Da 256 a 266 (0100/010A) serve per conversione da numero a stringa. Da 256 a 318 (0100/013E) serve per gli errori di Input da nastro; è chiamato simbolicamente BAD.

BUF 512/600 0200/0258
Buffer INPUT del sistema (89 byte).

LAT 601/610 0259/0262
Tabella KERNAL: numeri logici file attivi.

FAT 611/620 0263/026C
Tabella KERNAL: numeri periferiche dei file attivi.

SAT 621/630 026D/0276
Tabella KERNAL: indirizzi secondari file attivi.

KEYD 631/640 0277/0280
Buffer della tastiera, 10 byte gestiti in modo FIFO (First IN First Out - il primo che entra è il primo che esce). La coda è gestita dal puntatore in 198. Se 198 contiene 0, la coda è vuota.

MEMSTR 641/642 0281/0282
Puntatore parte bassa della memoria per Sistema Operativo; inizialmente contiene 2048.

MEMSIZ 643/644 0283/0284
Puntatore parte alta della memoria per Sistema Operativo; inizialmente contiene 40960.

TIMOUT 645 0285
Indicatore Timeout per IEEE.

COLOR 646 0286
Codice colore caratteri correnti.

GDCOL 647 0287
Colore dello sfondo sotto il cursore.

HIBASE 648 0288
Indirizzo pagina della mappa video; inizialmente contiene 4 (4x256=1024).

XMAX 649 0289
Dimensione del buffer della tastiera; inizialmente contiene 10.

RPTFLG 650 028A
Indicatore ripetizione automatica tasti premuti; 128 (80) per ON.

KOUNT 651 028B
Contatore velocità ripetizione.

DELAY 652 028C
Ritardo ripetizione.

SHFLAG 653 028D
Indicatore tasto premuto, distingue tra SHIFT, CBM, CTRL. 0 nessun tasto, 1 SHIFT, 2 CBM, 4 CTRL, 6 CBM+CTRL, 7 per tutti.

LSTSHF 654 028E
Ultima configurazione ottenuta con il tasto SHIFT.

KEYLOG 655/656 028F/0290
Vettore determinazione tabella tastiera.

MODE 657 0291
Indicatore: 0 (00) disabilita tasto SHIFT, 128 (80) abilita tasto SHIFT.

AUTODN 658 0292
Indicatore scrolling automatico verso il basso: 0 corrisponde a ON.

M51CTR 659 0293
RS232: immagine registro di controllo 6551.

M51CDR 660 0294
RS232: immagine registro di comando 6551.

M51AJB 661/662 0295/0296
RS232: BPS USA non standard (tempo/2-100).

RSSTAT 663 0297
RS232: Immagine registro di stato 6551.

BITNUM 664 0298

RS232: numero di bit che rimangono da inviare.

BAUDOF 665/666 0299/029A
RS232: velocità di trasmissione (baud rate); tempo per un bit completo in microsecondi.

RIDBE 667 029B
RS232: puntatore alla fine del buffer di Input.

RIDBS 668 029C
RS232: inizio (pagina) buffer di Input.

RODBS 669 029D
RS232: inizio (pagina) del buffer di Output.

RODBE 670 029E
RS232: puntatore alla fine del buffer di Output.

IRQTMP 671/672 029F/02A0
Contiene il vettore IRQ durante I/O nastro.

ENABL 673 02A1
Abilita RS232.

674 02A2
Controllo sensore cassetta durante I/O nastro.

675 02A3
Memoria temporanea durante lettura nastro.

676 02A4
Indicatore temporaneo IRQ durante lettura nastro.

677 02A5
Indicatore temporaneo indice linea.

678 02A6
Indicatore televisione: 0=NTSC, 1=PAL.

679/767 02A7/02FF
Non usato.

IERROR 768/769 0300/0301
Vettore per stampa messaggi errore Basic.

IMAIN	770/771	0302/0303

Vettore partenza a caldo del Basic.

ICRNCH	772/773	0304/0305

Vettore ingresso routine TOKEN.

IQPLOP	774/775	0306/0307

Vettore ingresso routine LIST.

IGONE	776/777	0308/0309

Vettore ingresso routine ricerca parole chiave.

IEVAL	778/779	030A/030B

Vettore ingresso valutazione Token.

SAREG	780	030C

Memorizzazione registro A 6502.

SXREG	781	030D

Memorizzazione registro X 6502.

SYREG	782	030E

Memorizzazione registro Y 6502.

SPREG	783	030F

Memorizzazione registro SP 6502.

USRPOK	784	0310

Salto funzione USR.

USRADD	785/786	0311/0312

Indirizzo USR LO/HI.

	787	0313

Non usato.

CINV	788/789	0314/0315

Vettore interrupt IRQ.

CBINV	790/791	0316/0317

Vettore interrupt BRK.

NMINV	792/793	0318/0319

Vettore interrupt non mascherabile.

IOPEN 794/795 031A/031B
Vettore ingresso routine KERNAL OPEN.

ICLOSE 796/797 031C/031D
Vettore ingresso routine KERNAL CLOSE.

ICHKIN 798/799 031E/031F
Vettore ingresso routine KERNAL CHKIN.

ICKOUT 800/801 0320/0321
Vettore ingresso routine KERNAL CHKOUT.

ICLRCH 802/803 0322/0323
Vettore ingresso routine KERNAL CLRCHN.

IBASIN 804/805 0324/0325
Vettore ingresso routine KERNAL CHRIN.

IBSOUT 806/807 0326/0327
Vettore ingresso routine KERNAL CHROUT.

ISTOP 808/809 0328/0329
Vettore ingresso routine KERNAL STOP.

IGETIN 810/811 032A/032B
Vettore ingresso routine KERNAL GETIN.

ICLALL 812/813 032C/032D
Vettore ingresso routine KERNAL CLALL.

USRCMD 814/815 032E/032F
Vettore ingresso routine definita dall'utente.

ILOAD 816/817 0330/0331
Vettore ingresso routine KERNAL LOAD.

ISAVE 818/819 0332/0333
Vettore ingresso routine KERNAL SAVE.

820/827 0334/033B
Non usato.

TBUFFR 828/1019 033C/03FB
Buffer cassetta, 192 byte.

	1020/1023	03FC/03FF

Non usato.

Partendo dalle indicazioni di questa tabella puoi fare diverse prove, ma devi essere armato di pazienza!

Riportiamo un esempio semplice che riguarda il byte 650: esso è di norma a zero e, di conseguenza, i tasti non hanno la funzione di ripetizione automatica, escludendo quei pochi che sono sempre abilitati alla ripetizione, come la barra di spazio. Segue il programma RIPETAUT, che inizialmente stampa il contenuto del byte 650, chiede una stringa in Input e la stampa, poi modifica il contenuto di 650, chiede una stringa in Input e la stampa. Nel secondo hai la ripetizione automatica per qualunque tasto premi e mantieni premuto.

```
1 REM RIPETAUT
5 POKE650,0
7 PRINT"BYTE 650: ";PEEK(650)
10 INPUTA$
20 PRINTA$
30 POKE650,128
35 PRINT"BYTE 650: ";PEEK(650)
40 INPUTA$
50 PRINTA$
55 POKE650,0:STOP
```

Segue il programma PUNT1 con il quale puoi stampare il contenuto di qualunque byte e puoi introdurre una stringa descrittiva per preparare delle tabelline a scopo di studio. Per uscire dal programma devi rispondere solo con RETURN alla richiesta della descrizione.

```
1 REM PUNT1
3 INPUT"RISULTATI SU STAMPANTE? (S/N) ";R$
4 PRINT:IFR$="N"THEN8
5 OPEN4,4
6 PRINT#4,"***CONTENUTO PUNTATORE A 1 BYTE***"
7 PRINT#4
8 D$="":INPUT"DESCRIZIONE PUNTATORE: ";D$
9 IFD$=""THEN40
10 INPUT"IND. BYTE: ";B
20 PRINTD$;SPC(2);"(";B;")";PEEK(B)
```

```
25 IFR$="N"THEN35
30 PRINT#4,D$;SPC(2);"(";B;")";PEEK(B)
35 PRINT#4:GOTO8
40 IFR$="S"THENCLOSE4
45 STOP
```

Segue il programma PUNT2 con il quale puoi stampare il contenuto dei puntatori a due byte accompagnandolo con una descrizione; anche questo è utile per studiare il comportamento del sistema.

```
1 REM PUNT2
3 INPUT"RISULTATI SU STAMPANTE? (S/N) ";R$
4 PRINT:IFR$="N"THEN8
5 OPEN4,4
6 PRINT#4,"***CONTENUTO PUNTATORI A 2 BYTE***"
7 PRINT#4
8 D$="":INPUT"DESCRIZIONE PUNTATORE: ";D$
9 IFD$=""THEN40
10 INPUT"IND. BYTE BASSO: ";B
15 INPUT"IND. BYTE ALTO: ";A
20 X=256*PEEK(A)+PEEK(B)
21 PRINTD$;SPC(2);"(";B;"/";A;")";X
25 IFR$="N"THEN35
30 PRINT#4,D$;SPC(2);"(";B;"/";A;")";X
31 PRINT#4
35 GOTO8
40 IFR$="S"THENCLOSE4
45 STOP
```

Infine ecco il programma ZONEMEM; esso serve per stampare, byte dopo byte, il contenuto di zone di memoria, premettendo una breve frase. A scopo di esempio l'abbiamo fatto girare per stampare il contenuto dei 256 byte dell'area STACK.

```
1 REM ZONEMEM
2 INPUT"RISULTATI SU STAMPANTE? (S/N) ";R$
3 PRINT:IFR$="N"THEN5
4 OPEN4,4
5 D$="":INPUT"DESCRIZIONE ZONA: ";D$
7 IFD$=""THEN40
10 INPUT"IND. PRIMO BYTE: ";B
```

```
15 INPUT"IND. SECONDO BYTE: ";A
20 PRINTD$;SPC(2);"(";B;"/";A;")"
21 C=0:FORJ=BTOA:PRINTPEEK(J);"  ";
22 C=C+1:IFC=6THENPRINT:C=0
23 NEXTJ:PRINT
25 IFR$="N"THEN35
26 PRINT#4,"***CONTENUTO ZONE DI MEMORIA***"
27 PRINT#4
30 PRINT#4,D$;SPC(2);"(";B;"/";A;")":PRINT#4
31 C=0:FORJ=BTOA:PRINT#4,PEEK(J);
32 C=C+1:IFC=8THENPRINT#4:C=0
33 NEXTJ:PRINT#4
35 GOTO5
40 IFR$="S"THENCLOSE4
45 STOP
```

```
***CONTENUTO ZONE DI MEMORIA***

STACK SISTEMA  ( 256 / 511 )

 32  51  49  0  48  48  48  48
 48  48  48  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  251  255  0  0  255  255
 232  0  255  255  0  0  252  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  200  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  251  255  0  0  255  255
```

```
232  0  255  255  0  0  255  255
0  0  255  255  0  0  255  255
0  0  255  255  0  0  255  255
0  0  255  255  0  125  125  125
234  0  1  0  125  14  1  14
188  97  125  189  172  176  183  188
3  184  21  184  5  0  226  175
179  173  0  183  170  130  170  233
167  129  236  9  129  128  0  0
0  1  137  127  128  0  0  31
0  9  127  167  121  166  156  44
```

L'analisi del contenuto dell'area stack, non è molto semplice, bisognerebbe conoscere il funzionamento delle routine del sistema. Per esempio, nell'area stack sono memorizzate le indicazioni necessarie per gestire i cicli FOR; infatti tra le variabili Basic si trova la variabile di controllo del ciclo come una normale variabile floating-point. Il sistema per gestire il ciclo FOR ha bisogno del valore dello STEP, del valore finale per il contatore, dell'indirizzo di memoria dove iniziare ogni ciclo e dell'indirizzo di memoria dove andare alla fine del ciclo.

1.5 CONFIGURAZIONI

Il COMMODORE 64 dispone di 64K di memoria, più altri 20K di memoria, diciamo, ombra. Di fatto il calcolatore lavora indirizzando solo 64K, ma è possibile scambiare tra loro due blocchi da 8K e uno da 4K (8+8+4=20). Facciamo riferimento alla Figura 8.2. In essa la memoria è rappresentata da un rettangolo suddiviso in blocchi; a sinistra sono scritti gli indirizzi decimali e, tra parentesi, esadecimali di inizio e fine di ogni blocco. All'interno di ogni blocco compaiono delle diciture che specificano il possibile uso del blocco. Tre blocchi sono contrassegnati da un asterisco a destra; essi sono i blocchi "doppi". Essi sono effettivamente doppi, cioè esistono ambedue i blocchi, corrispondono agli stessi indirizzi, ma sono attivi alternativamente, o l'uno o l'altro.

Per quanto riguarda il blocco di 8K che va da 32768 a 40959 (8000-9FFF) il discorso è diverso; esso è presente nel calcolatore come RAM, ma, usando la porta di espansione che comunica con l'esterno sul retro del calcolatore, si può inserire una ROM, che va a sostituirsi agli stessi indirizzi della RAM escludendola.

Nella Figura 1.6 del Capitolo 1 del primo Volume si è mostrata la configurazione della memoria al momento dell'accensione del calcolatore; in essa i 4K da 53248 a

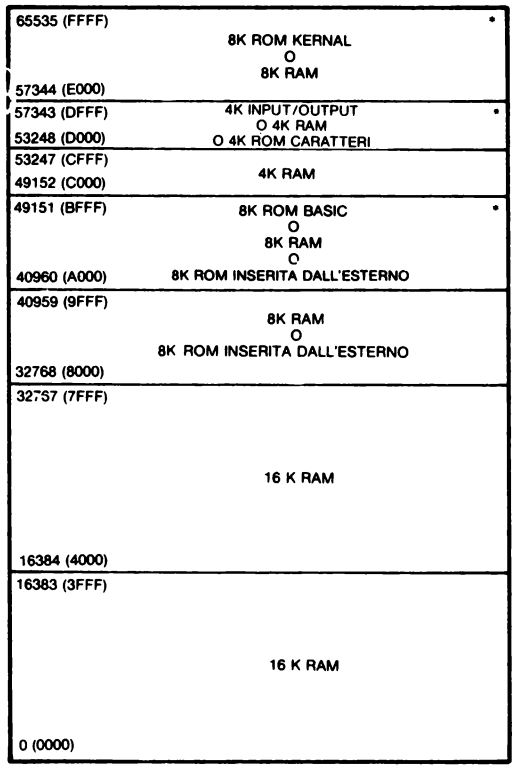


Figura 1.2 Mappa della memoria

57343 (D000-DFFF) sono stati indicati come I/O. Poi abbiamo visto che da quegli stessi indirizzi si possono andare a prelevare i caratteri in ROM, usando però un particolare accorgimento nel programma (DISECAR). In effetti, con la tecnica della memoria ombra, è possibile associare a questo spazio di 4K indirizzi o una zona di memoria ROM o gli indirizzi dei dispositivi di I/O utilizzati dal sistema.
Vediamo ora più da vicino come si stabilisce la configurazione della memoria.
I primi 2 indirizzi di memoria (∅ e 1) corrispondono a 2 registri interni della CPU che servono appunto per definire la configurazione della memoria attiva.
All'indirizzo ∅ si trova il DATA DIRECTION REGISTER (D6510), all'indirizzo 1 si trova il REGISTRO DI I/O (R6510).
A questi 2 registri fanno capo 6 linee della CPU che, opportunamente programmate come vedremo tra breve, attivano o disattivano diverse zone di memoria.
Al momento dell'accensione del calcolatore il D6510 contiene 47 e il R6510 contiene 55. Vediamo il significato dei bit di questi importanti registri.

I bit del registro DATA DIRECTION assegnano alle corrispondenti linee la

funzione di INPUT o OUTPUT; precisamente 1 significa OUTPUT e 0 significa INPUT. La configurazione iniziale 47 corrisponde alla seguente disposizione di 1 e 0:

	bit	7	6	5	4	3	2	1	0
D6510		0	0	1	0	1	1	1	1

I bit di posizione 6 e 7 non sono significativi; gli altri assegnano la funzione di Output, salvo il bit di posizione 4 che assegna la funzione di Input.

Le 6 linee che fanno capo ai registri D6510 e R6510 hanno il seguente significato:

NOME	POS.BIT	DIREZIONE	FUNZIONE
LORAM	0	Output	Controllo zona di memoria 40960/49151 (A000/BFFF)
HIRAM	1	Output	Controllo zona di memoria 57344/65535 (E000/FFFF)
CHAREN	2	Output	Controllo zona di memoria 53248/57343 (D000/DFFF)
	3	Output	Linea scrittura cassetta
	4	Input	Switch interrutori cassetta
	5	Output	Controllo motore cassetta

All'inizio questa porta contiene 55, cioè la seguente configurazione di bit:

	bit	7	6	5	4	3	2	1	0
R6510		0	0	1	1	0	1	1	1

e i bit 0, 1 e 2 al valore 1 danno luogo alla configurazione riportata in Figura 1.6.

Vediamo il significato delle linee della porta 1.

La LORAM è la linea che controlla la presenza o meno degli 8K ROM del BASIC nello spazio degli indirizzi del microprocessore. La ROM è attiva se la linea è programmata alta (HIGH). Quando la linea è programmata bassa (LOW) la ROM sparisce ed al suo posto risultano attivi 8K di RAM.

La HIRAM è la linea che controlla la presenza o meno degli 8K di ROM KERNAL

(Sistema Operativo) nello spazio degli indirizzi del microprocessore. La ROM è attiva quando la linea è programmata alta (HIGH). Quando la linea è programmata bassa (LOW) la ROM sparisce e al suo posto risultano attivi 8K di RAM; contemporaneamente spariscono, ovviamente, anche gli 8K di ROM BASIC.

La CHAREN controlla la presenza dei 4K di ROM caratteri quando è al valore 0, mentre quando è al valore 1 sono attivi o 4K RAM o 4K I/O, in accordo con LORAM e HIRAM. Lo scambio tra questi due blocchi di 4K di memoria avviene tutte le volte che serve; ne abbiamo visto un esempio nel programma DISECAR del Capitolo 7.

Lo spazio degli indirizzi di 4K dedicato agli I/O è il seguente:

53248/54271	(D000/D3FF)	Controllo Video VIC (1024 byte)
54272/55295	(D400/D7FF)	Sintetizzatore Suoni SID (1024 byte)
55296/56319	(D800/DBFF)	RAM colore (1024 byte)
56320/56575	(DC00/DCFF)	Tastiera CIA1 (256 byte)
56576/56831	(DD00/DDFF)	Bus Seriale, Porta Utente RS-232 (256 byte)
56832/57087	(DE00/DEFF)	Slot # 1 (usato per CP/M (256 byte)
57088/57343	(DF00/DFFF)	Slot # 2 (disco) (256 byte)

Nel determinare la configurazione di memoria attiva giocano un ruolo le linee della porta 1 descritte e, inoltre, i PIN 8 e 9 della porta di espansione; il PIN 8 prende il nome di GAME e il PIN 9 di EXROM. Nella configurazione iniziale con la porta di espansione non utilizzata, GAME e EXROM sono a 1

Riportiamo i possibili utilizzi della memoria che risultano dalla situazione delle linee: LORAM, HIRAM, GAME e EXPROM, con breve commento. (Una linea è "alta" quando è a 1 e "bassa" quando è a 0).

LORAM=1, HIRAM=0, GAME=1, EXROM=qualunque

0/16383	(0000/3FFF)	16K RAM
16384/32767	(4000/7FFF)	16K RAM
32768/49151	(8000/BFFF)	16K RAM
49152/53247	(C000/CFFF)	4K RAM
53248/57343	(D000/DFFF)	4K I/O
57344/65535	(E000/FFFF)	8K RAM

sono disponibili 60K di RAM e 4K di I/O, ma è assente sia il Basic che il Sistema Operativo.

LORAM=1, HIRAM=0, GAME=0, EXROM=0

dà la stessa disposizione di cui sopra, ma è inibito l'accesso alla mappa dei caratteri in ROM.

LORAM=0, HIRAM=1, GAME=1, EXROM=qualunque

0/16383	(0000/3FFF)	16K RAM
16384/32767	(4000/7FFF)	16K RAM
32768/49151	(8000/BFFF)	16K RAM
49152/53247	(C000/CFFF)	4K RAM
53248/57343	(D000/DFFF)	4K I/O
57344/65535	(E000/FFFF)	8K ROM

dà 52K di RAM consecutivi, 4K di I/O e le routine KERNAL; è una confuigurazione adatta ad essere usata con altri linguaggi e altri Sistemi Operativi.

LORAM=0, HIRAM=0, GAME=1/qualunque, EXROM=qualunque/0

dà 64K di RAM, ma manca l'I/O e deve essere immesso tutte le volte che serve.

LORAM=1, HIRAM=1, GAME=1, EXROM=0

è la configurazione standard con una ROM esterna (in generale l'espansione Basic) posizionata da 32768 a 40959 (8000/9FFF).

LORAM=0, HIRAM=1, GAME=0, EXROM=0

corrisponde alla configurazione base dove il BASIC è stato sostituito da una ROM di 8K inserita dall'esterno; serve per programmi speciali che lavorano senza il Basic.

LORAM=1, HIRAM=1, GAME=0, EXROM=0

corrisponde alla configurazione base dove è stata inserita una ROM da 16K che si situa da 32768 a 49151 (8000/BFFF) e si sostituisce al BASIC e ad 8K di RAM. Serve per applicazioni speciali tipo Word Processor o altro.

Quando non si cita CHAREN, è possibile porla a zero e attivare la ROM dei caratteri.

1.6 ASSEGNAZIONI MEMORIA PER I/O

Riportiamo l'assegnazione degli indirizzi di memoria dei 4K da 53248 a 57343 (D000H-DFFFH) per l'I/O, quando questa zona è attiva come I/O. Nel seguito non facciamo seguire H ai numeri in esadecimali che riportiamo tra parentesi.

CONTROLLO VIDEO 53248/53294 (D000/D02E)
(VIC) MOS 6566

53248/53249	(D000/D001)	Pos. X/Y Sprite 0
53250/53251	(D002/D003)	Pos. X/Y Sprite 1
53252/53253	(D004/D005)	Pos. X/Y Sprite 2
53254/53255	(D006/D007)	Pos. X/Y Sprite 3
53256/53257	(D008/D009)	Pos. X/Y Sprite 4
53258/53259	(D00A/D00B)	Pos. X/Y Sprite 5
53260/53261	(D00C/D00D)	Pos. X/Y Sprite 6
53262/53263	(D00E/D00F)	Pos. X/Y Sprite 7
53264	(D010)	Pos. X Sprite 0/7, byte alto (MSB)
53265	(D011)	REGISTRO CONTROLLO

BIT 7: comparatore quadro (bit 8, vedi anche 53266)
BIT 6: modo testo colore esteso, 1 abilita
BIT 5: modo Bit-Map, 1 abilita
BIT 4: riempie lo schermo con il colore del bordo
BIT 3: seleziona 24/25 righe testo, 1 per 25 righe
BIT 2/0: scorrimento rallentato fino alla posizione Y del punto
53266 (D012) Lettura/Scrittura valore RASTER per confronto IRQ
53267 (D013) Latch Pos. X penna ottica
53268 (D014) Latch Pos. Y penna ottica
53269 (D015) Abilita animazione video, 1 abilita
53270 (D016) REGISTRO CONTROLLO
BIT 7/6: non usati
BIT 5: DEVE ESSERE SEMPRE ZERO
BIT 4: modo Multicolor, 1 abilita
BIT 3: seleziona 38/40 colonne video, 1 abilita
BIT 2/0: Scorrimento rallentato fino alla posizione X
53271 (D017) espansione verticale Sprite 0/7, 2 volte Y
53272 (D018) REGISTRO CONTROLLO MEMORIA
BIT 7/4: indirizzo base della matrice video all'interno del VIC
BIT 3/0: indirizzo base di un punto/carattere al'interno del VIC
53273 (D019) REGISTRO INDICATORE INTERRUZIONI, 1 per IRQ verificato
BIT 7: 1 per qualunque IRQ
BIT 3: indicatore IRQ penna ottica
BIT 2: indicatore IRQ collisione tra Sprite

BIT 1: indicatore IR collisione Sprite/sfondo
BIT 0: indicatore IRQ confronto RASTER
53274 (D01A) REGISTRO MASCHERA IRQ, 1 abilita interruzione
53275 (D01B) priorità Sprite/sfondo,1 per Sprite
53276 (D01C) modo Multicolor per Sprite 0/7, 1 abilita
53277 (D01D) espansione orizzontale Sprite 0/7, 2 volte X
53278 (D01E) riconoscimento contatto tra due Sprite
53279 (D01F) riconoscimento contatto Sprite/sfondo
53280 (D020) colore bordo
53281 (D021) colore sfondo 0
53282 (D022) colore sfondo 1
53283 (D023) colore sfondo 2
53284 (D024) colore sfondo 3
53285 (D025) registro 0 Sprite Multicolor
53286 (D026) registro 1 Sprite Multicolor
53287/53294 (D027/D02E) colore Sprite da 0 a 7

SINTETIZZATORE SUONI 54272/54300 (D400/D41C)
(SID) MOS 6581

54272 (D400) controllo frequenza voce 1 (byte LO)
54273 (D401) controllo frequenza voce 1 (byte HI)
54274 (D402) ampiezza onda quadra (byte LO)
54275 (D403) ampiezza onda quadra (byte HI)
BIT 7/4: non usati
BIT 3/0: semibyte HI
54276 (D404) REGISTRO CONTROLLO VOCE 1
BIT 7: seleziona forma onda Rumore, 1 abilitato
BIT 6: seleziona onda quadra, 1 abilitato
BIT 5: seleziona onda "dente di sega", 1 abilitato
BIT 4: seleziona onda triangolare, 1 abilitato
BIT 3: di controllo,1 disabilita oscillatore 1
BIT 2: modulatore anello oscillatore 1 con uscita oscillatore 3, 1 abilitato
BIT 1: sincronizzazione frequenza oscillatore1/oscillatore 3, 1 abilitato
BIT 0: controllo porta, 1 attiva Attacco/Decadimento/Sostegno, 0 attiva rilascio
54277 (D405) generatore inviluppo 1 per Attacco/Decadimento ciclo
BIT 7/4: seleziona ciclo Attacco durata 0/15
BIT 3/0: seleziona ciclo Decadimento durata 0/15
54278 (D406) generatore inviluppo 1 per Sostegno/Rilascio
BIT 7/4: seleziona ciclo Sostegno durata 0/15
BIT 3/0: seleziona ciclo Rilascio durata 0/15
54279/54283 (D407/D40B) come 54272/54276, ma per voce 2
54284/54285 (D40C/D40D) come 54277/54278, ma per generatore 2 inviluppo

54286/54290 (D40E/D412) come 54272/54276, ma per voce 3
54291/54292 (D413/D414) come 54277/54278, ma per generatore 3 inviluppo
54293 (D415) frequenza taglio del filtro, semibyte LO (BIT 2/0)
54294 (D416) frequenza taglio del filtro, byte HI
54295 (D417) controllo risonanza filtro/ingresso voce
BIT 7/4: selezione risonanza filtro 0/15
BIT 3: ingresso esterno filtro, 1 per SI, 0 per NO
BIT 2: uscita filtro voce 3, 1 per SI, 0 per NO
BIT 1: uscita filtro voce 2, 1 per SI, 0 per NO
BIT 0: uscita filtro voce 1, 1 per SI, 0 per NO
54296 (D418) selezione modo e volume filtro
BIT 7: taglio uscita voce 3, 1 per OFF, 0 per ON
BIT 6: selezione filtro passa-alto, 1 per ON
BIT 5: selezione filtro passa-banda, 1 per ON
BIT 4: seleziona filtro passa-basso, 1 per ON
BIT 3/0: selezione volume uscita 0/15
54297 (D419) convertitore analogico/digitale Paddle 1, 0/255
54298 (D41A) convertitore analogico/digitale Paddle 2, 0/255
54299 (D41B) generatore numeri a caso oscillatore 3
54300 (D41C) uscita generatore inviluppo 3

MAPPA COLORE 55296/56319 (D800/DBFF)

contiene i colori corrispondenti alle posizioni della MAPPA VIDEO

ADATTATORE INTERFACCIA COMPLESSA CIA 1 56320/56335 (DC00/DC0F)
MOS 6526

56320 (DC00) porta A dati (tastiera, joystick, paddle)
BIT 7/0: scrive valori colonna scansione tastiera
BIT 7/6: legge paddle, 01 per porta A, 10 per porta B
BIT 4: pulsante sparo joystick A, 1 per fuoco
BIT 3/2: pulsanti sparo paddle
BIT 3/0: direzione joystick A, 0/15
56321 (DC01) porta B dati: (tastiera, joystick, paddle)
BIT 7/0: legge valori colonna scansione tastiera
BIT 7: timer B, uscita impulso
BIT 6: timer A, uscita impulso
BIT 4: pulsante sparo joystick 1, 1 per fuoco
BIT 3/2: pulsanti sparo paddle
BIT 3/0: direzione joystick 1
56322 (DC02) REGISTRO DIREZIONE DATI Porta A

56323 (DC03) REGISTRO DIREZIONE DATI Porta B
56324 (DC04) timer A, byte LO
56325 (DC05) timer A, byte HI
56326 (DC06) timer B, byte LO
56327 (DC07) timer B, byte HI
56328 (DC08) orologio, contatore decimi secondo
56329 (DC09) orologio, contatore secondi
56330 (DC0A) orologio, contatore minuti
56331 (DC0B) orologio, contatore ore BIT 6/0, BIT 7 indicatore AM/PM
56332 (DC0C) buffer dati I/O seriale sincrono
56333 (DC0D) REGISTRO CONTROLLO INTERRUZIONI CIA (lettura IRQ/ maschera scrittura)
BIT 7:Indicatore IRQ, 1 segnala interruzione
BIT 4: indicatore 1 IRQ, lettura cassetta/ input SRQ bus seriale
BIT 3: interruzione porta seriale
BIT 2: interruzione orologio
BIT 1: interrruzione timer B
BIT 0: interruzione timer
56334 (DC0E) REGISTRO A CONTROLLO CIA
BIT 7: frequenza orologio, 1 per 50 Hz, 0 per 60 Hz
BIT 6: modo porta seriale, 1 per Output, 0 per Input
BIT 5: contatore timer A, 1 per segnali CNT, 0 per clock 2 sistema
BIT 4: caricamento forzato timer A, 1 per SI
BIT 3: modo funzionamento timer A, 1 per monostabile, 0 per continuo
BIT 2: modo uscita timer A per PB6, 1 per bistabile, 0 per pulsazione
BIT 1: uscita timer A per PB6, 1 per SI, 0 per NO
BIT 0: flag timer a, 1 attiva, 0 ferma
56335 (DC0F) REGISTRO B CONTROLLO CIA
BIT 7: predispone allarme/TOD clock 1, 1 per allarme, 0 per TOD
BIT 6/5: selezione modo timer B:
00 = conteggio pulsazioni clock 02
01 = conteggio transizioni positive CNT
10 = conteggio pulsazioni underflow timer A
11 = conteggio pulsazioni underflow timer A mentre si mantiene positivo CNT
BIT 4/0: come gli stessi bit del registro di controllo A, ma per il timer B

ADATTATORE INTERFACCIA COMPLESSA CIA 2
56576/56591 (DD00/DD0F)
MOS 6526

56576 (DD00) porta dati A (bus seriale, RS-232, controllo memoria VIC)
BIT 7: ingresso bus seriale
BIT 6: ingresso impulsi clock bus seriale

BIT 5: uscita bus seriale
BIT 4: uscita impulsi clock bus seriale
BIT 3: uscita segnale ATN bus seriale
BIT 2: uscita dati RS-232 (porta utente)
BIT 1/0: selezione banco memoria sistema del chip VIC, valore per difetto 11
56577 (DD01) porta dati B (porta utente, RS-232)
BIT 7: Data Set Ready
BIT 6: Clear to Send
BIT 5: utente
BIT 4: Carrier Detect
BIT 3: Ring Indicator
BIT 2: Data Terminal Ready
BIT 1: Request to Send
BIT 0: Received Data
56578 (DD02) REGISTRO DIREZIONE DATI PORTA A
56579 (DD03) REGISTRO DIREZION DATI PORTA B
56580 (DD04) timer A, byte LO
56581 (DD05) timer A, byte HI
56582 (DD06) timer B, byte LO
56583 (DD07) timer B, byte HI
56584 (DD08) orologio, decimi secondo
56585 (DD09) orologio, secondi
56586 (DD0A) orologio, minuti
56587 (DD0B) orologio, ore BIT 6/0, flag AM/PM BIT 7
56588 (DD0C) buffer dati seriali
56589 (DD0D) REGISTRO CONTROLLO INTERRUZIONI CIA (lettura NMI/scrittura maschera)
BIT 7: indicatore NMI, 1 per avvenuto NMI
BIT 4: indicatore 1 NMI, dati input utente/RS-232
BIT 3: interruzione porta seriale
BIT 1: interruzione timer B
BIT 0: interruzione timer A
56590 (DD0E) REGISTRO A CONTROLLO CIA
disposizione BIT come 56334
56591 (DD0F) REGISTRO B CONTROLLO CIA
disposizione BIT come 56335

CAPITOLO 2

IL SISTEMA OPERATIVO GEOS

2.1 IL SISTEMA OPERATIVO GEOS

Il sistema operativo GEOS (Graphic Environment Operating System) fornisce un *ambiente di lavoro che rende molto semplice e intuitivo l'utilizzo del calcolatore*. Esso presenta un *menu a icone,* cioè a immagini, e riporta costantemente sul video i comandi da utilizzare per poter lavorare.
Per *attivare* un comando è sufficiente *spostare la freccia-indicatore* sul comando o sull'icona e *premere un pulsante una o più volte*. In seguito alla scelta di un comando compaiono sullo schermo, sovrapponendosi all'immagine preesistente, *finestre video con l'elenco delle opzioni possibili (pull down-menu);* la scelta viene effettuata sempre spostando la freccia e azionando il pulsante.

Noi abbiamo provato la versione 1.2 del sistema operativo GEOS, usando un dischetto registrato su ambedue le facce, utilizzandolo però una faccia alla volta, sia con l'unità 1541 che con l'unità 1571.

Sulla prima faccia del floppy sono registrati i seguenti file:

blocchi	*nome file*	*tipo*	*utilizzo*
1	GEOS	PRG<	caricatore sistema operativo
6	GEOS BOOT	PRG<	lancio sistema
85	GEOS KERNAL	USR<	routine sistema
72	DESK TOP	USR<	gestione sistema
119	GEOPAINT	USR<	pacchetto grafico
88	GEOWRITE	USR<	pacchetto scrittura
16	BACKUP	PRG<	copia dischi
22	PREFERENCE MGR	USR	selezione caratteristiche ambiente di lavoro
13	ALARM CLOCK	USR	orologio e sveglia
23	PHOTO MANAGER	USR	gestione album di disegni
20	TEXT MANAGER	USR	gestione album di testi
16	CALCULATOR	USR	calcolatrice
17	NOTE PAD	USR	taccuino appunti
3	ePSON mx-80	USR	driver printer EPSON MX-80
26	cALIFORNIA	USR	descrizione caratteri
23	cORY	USR	descrizione caratteri
13	dWINELLE	USR	descrizione caratteri
34	rOMA	USR	descrizione caratteri
40	uNIVERSITY	USR	descrizione caratteri

4	mps-801	USR	driver printer MPS-801
4	mps-1000	USR	driver printer mps-1000
4	cOMM. cOMPAT.	USR	driver printer Comm. compat.
3	bLUEcHIP m120	USR	driver BlueChip m120
3	c.iTOH 8510	USR	driver C.Itoh 8510
3	joystick	USR	input driver joystick

I primi tre file contengono il sistema GEOS; il file DESK TOP contiene i quadri video utilizzati dal sistema. Il carattere '<', che compare dopo il tipo del file nella lista della directory del floppy, significa che il file è protetto, cioè che non può essere cancellato.

Sulla seconda faccia del floppy sono registrati i seguenti file:

blocchi	*nome file*	*tipo*	*utilizzo*
22	fONTkNOX	USR	descrizione caratteri
20	bUBBLE	USR	descrizione caratteri
28	bOALT	USR	descrizione caratteri
32	eLMWOOD	USR	descrizione caratteri
20	sUPERB	USR	descrizione caratteri
34	dURANT	USR	descrizione caratteri
24	tILDEN	USR	descrizione caratteri
22	tOLMAN	USR	descrizione caratteri
9	eVANS	USR	descrizione caratteri
17	hARMON	USR	descrizione caratteri
32	hEARST	USR	descrizione caratteri
15	lEcONTE	USR	descrizione caratteri
22	mYKONOS	USR	descrizione caratteri
26	oRMAND	USR	descrizione caratteri
21	pUTNAM	USR	descrizione caratteri
13	sTADIUM	USR	descrizione caratteri
9	lolp	SEQ	file non GEOS
22	bOWDITCH	USR	descrizione caratteri
11	tELEGRAPH	USR	descrizione caratteri
24	cHANNING	USR	descrizione caratteri
11	bRENNENS	USR	descrizione caratteri
3	ePSON fx-80	USR	driver printer Epson FX-80

Per utilizzare il GEOS è necessario servirsi di un *joystick* o di un *mouse* collegato alla porta 1.
Per far partire il sistema devi procedere in questo modo:
- accendere il video, il calcolatore, l'unità a disco, se vuoi, la stampante, e inserire o un joystick o un mouse nella porta 1;

- inserire il floppy del sistema nell'unità con la faccia 1 verso l'alto;
- scrivere: LOAD "GEOS",8 seguito da RETURN per caricare il sistema e RUN per farlo partire, oppure scrivere: LOAD"GEOS,8,1 seguito da RETURN per caricare e far partire il GEOS.

Dopo il *booting* del sistema vedrai comparire sul video il primo menu, che rappresenta il tuo *tavolo da lavoro*, riportato in figura 2.1.

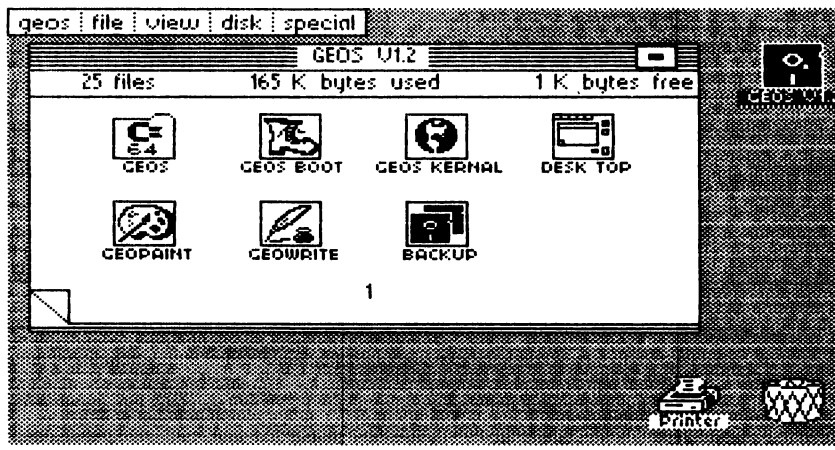


Figura 2.1 Tavolo da lavoro del GEOS (DESKTOP)

In alto a sinistra sul video compare la freccia-indicatore, che si sposta utilizzando il joystick o il mouse, e serve per selezionare i comandi. Il cambio del colore dello sfondo indica che è stato selezionato un comando.
La prima striscia chiara in alto a sinistra mostra i comandi:
GEOS, FILE, VIEW, DISK, SPECIAL. Portando la freccia su un comando e premendo il bottone del fuoco (l'unico sul joystick e quello di sinistra sul mouse), il comando viene attivato e compare in corrispondenza una finestra sovrapposta con le opzioni possibili (pull down-menu). Per scegliere un'opzione devi *spostare la freccia verso il basso* e premere il bottone del fuoco. Se sposti la freccia in altre direzioni scompare il menu delle opzioni e il comando si disattiva.
Sulla destra del video in alto compare la sagoma di un floppy in scuro, perchè è attivo, e con la dicitura 'GEOS V1.2', a indicare che si tratta del disco sistema.
La parte centrale del video è occupata da un taccuino per note. Nella parte superiore del taccuino è riportata l'intestazione *GEOS V1.2* e, di fianco, la sagoma della finestra dell'unità disco. Nella riga sottostante sono riportate alcune indicazioni sullo stato del floppy attivo, e precisamente:
-numero dei file presenti nella directory;

-numero dei Kbyte occupati e numero dei Kbyte liberi sul floppy.
La parte centrale del taccuino riporta le icone di al massimo 8 dei file presenti sul floppy, con il nome dei file. La pagina del taccuino è numerata con 1, e l'angolo ripiegato in basso a sinistra dà la possibilità di *sfogliare* il taccuino. Per sfogliare il taccuino devi portare la freccia sull'angolo in basso a sinistra e premere il bottone del fuoco; agendo sull'angolo del foglio sottostante si passa alla pagina precedente, mentre agendo sulla parte ripiegata si passa alla pagina successiva. Le pagine visualizzabili sono in tutto 8.
Nella parte in basso a destra del video appaiono le sagome della stampante e del cestino della carta straccia.

COSA SI PUÒ FARE

Il GEOS ti consente di:
- gestire i file presenti sul floppy, utilizzando anche una seconda unità a disco;
- utilizzare una stampante;
- utilizzare i pacchetti GEOPAINT e GEOWRITE già presenti sul floppy;
- accedere a programmi BASIC scritti da te e utilizzarli;
- utilizzare *accessori* disponibili, come *calcolatrice, taccuino per appunti e altro* che vedremo.

MODALITÀ DI COMANDO

Per impartire i comandi devi spostare la freccia sulla parola o sull'icona scelta e premere il bottone del fuoco; vedrai cambiare il colore dello sfondo.
Premendo due volte di seguito il bottone del fuoco, dopo aver portato la freccia su un'icona che rappresenta un programma, vedrai cambiare colore allo sfondo del nome del file, sentirai che viene caricato il programma da disco, e poi fatto partire.
Premendo due volte il bottone del fuoco con un intervallo di almeno un secondo, dopo aver portato la freccia su un'icona che rappresenta un programma, vedrai comparire un'ulteriore sagoma dell'icona in colore diverso; tale sagoma può essere trascinata sul video spostando la freccia e depositata in un altro posto premendo il bottone del fuoco.
Se porti la freccia su un'icona relativa ad un file e premi il bottone del fuoco essa viene selezionata e cambia colore; se a questo punto attivi uno dei comandi superiori il menu secondario che compare nella finestra sovrapposta riguarda il file selezionato.
Durante il colloquio, che si svolge per l'esecuzione di alcuni comandi, compare un menu secondario in una finestra sovrapposta in posizione centrale; in tale riquadro viene chiesto di eseguire, con la solita procedura di spostamento della freccia e pressione del bottone del fuoco, una azione come confermare (OK) o annullare il comando (CANCEL) o abbandonare (QUIT).

OPERAZIONI PRELIMINARI CONSIGLIATE

Il floppy del sistema contiene molti file e rimane poco spazio libero; per poter lavorare occorre spazio sul floppy; in conseguenza ti consigliamo di procurarti alcuni floppy e di procedere in questo modo.
Per prima cosa ti conviene eseguire una copia di ambedue le facce del floppy. La copia della prima faccia non può essere utilizzata per lanciare il sistema, ma serve per sicurezza, se si danneggia qualche file del floppy originale, e per lavorare.
Infatti una volta ottenute le copie delle due facce del floppy conviene lanciare il GEOS utilizzando il floppy originale, poi togliere il dischetto, inserire la copia, portare la freccia sulla sagoma del floppy in alto a sinistra e premere il bottone del fuoco; in tale modo si apre il nuovo dischetto inserito e sul taccuino compare una parte della sua directory. Il dischetto copia può essere utilizzato senza proteggere la finestra con l'etichetta e quindi può essere anche scritto.

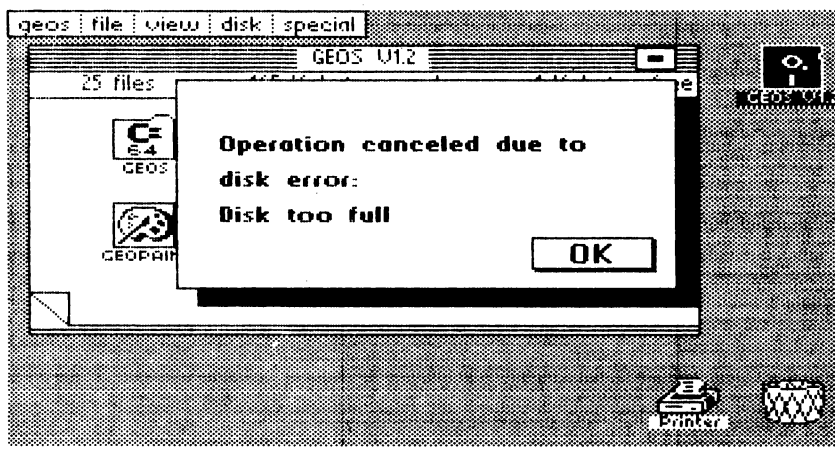


Figura 2.2 Segnalazione di disco pieno

Inoltre ti conviene eseguire diverse copie del floppy originale e, dopo averle eseguite, eliminare cancellandoli i file che non ti servono per le tue particolari applicazioni, così ti crei spazio sul dischetto per poter lavorare.

COME SI COPIA UN FLOPPY

Partire dal *desktop* iniziale. Il disco sorgente (quello da copiare) deve essere protetto chiudendo la finestrella laterale con l'apposita etichetta.
1) Portare la freccia sull'icona del programma BACKUP e premere due volte il

pulsante del fuoco per selezionare il programma e farlo partire.
2) Sul video compare il messaggio:

DISK BACKUP/RESTORE UTILITY
INSERT DESTINATION DISK TO BE FORMATTED AND ENTER 'F' TO FORMAT, OR ENTER 'Q' TO QUIT (F/Q)?

e devi sostituire il disco del sistema con un disco (cancellabile) da utilizzare per la copia, che verrà formattato e rispondere con 'F'. Se vuoi rinunciare all'operazione rispondi con 'Q'.
3) Terminata la formattazione del disco destinazione compare sul video il messaggio:

INSERT SOURCE DISK, AND ENTER 'C' TO COPY (C)?

Dopo aver scambiato il disco e risposto con 'C', compare il messaggio:

READING SOURCE DISK ...

e viene caricata in memoria una parte del dischetto. A caricamento avvenuto compare il messaggio:

PLEASE INSERT DESTINATION DISK

e devi inserire il nuovo disco al posto del vecchio, *senza premere alcun tasto*, immediatamente dopo la chiusura dello sportellino compare il messaggio:

WRITING DESTINATION DISK ...

e l'operazione procede.
Dopo poco compare il messaggio:

PLEASE INSERT SOURCE DISK

e il colloquio procede richiedendo lo scambio dei floppy quando è necessario. Al termine delle operazioni compare il messaggio:

BACKUP COMPLETE INSERT GEOS BOOT DISK AND PRESS RESTORE

e dopo l'inserimento del disco del sistema ricompare il desktop iniziale del GEOS.
L'operazione di copiatura di un floppy richiede circa 4 minuti.
Il programma BACKUP può essere lanciato anche da BASIC.

COME SI APRE E SI CHIUDE UN FLOPPY

Per aprire portare la freccia sull'icona del floppy fuori a destra del taccuino e premere una volta il bottone del fuoco.
L'effetto di questa operazione è di far comparire il menu a icone della prima pagina della directory del disco che è inserito, dopo aver attivato il floppy, e evidenziato il nome del floppy sotto la relativa icona. Se tale operazione si compie dopo uno scambio di floppy, il precedente viene chiuso.

Per chiudere portare la freccia sulla finestrella dell'unità disegnata a destra nella parte alta del taccuino e premere il bottone del fuoco.
Tale operazione, eseguita con un floppy inserito e attivato, fa chiudere il floppy evidenziando un punto interrogativo sulla relativa icona, scomparire il menu a icone e il nome del floppy.

Queste operazioni possono anche essere ottenute utilizzando i comandi del menu DISK che vedremo più avanti.

COME SI CANCELLA UN FILE

Partire dal *desktop* iniziale.
1) Far comparire la pagina che contiene l'icona del file agendo sull'angolo ripiegato del taccuino.
2) Portare la freccia sull'icona del file da cancellare e premere il bottone del fuoco una volta per selezionarlo e una seconda volta per far comparire la sagoma di colore diverso.
3) Trascinare la sagoma spostando la freccia sopra il cestino della carta straccia e premere una volta il bottone del fuoco.
Vedrai scomparire l'icona dal menu e vedrai modificare il numero che rappresenta i Kbyte liberi del floppy.

COME SI COPIA UN FILE

Partire *dal desktop* iniziale. Le operazioni differiscono a seconda che siano collegate due o una unità.

Copia di un file tra floppy diversi disponendo di una sola unità a disco.
1) Evidenziare sul taccuino l'icona del file da copiare che sta sul floppy attivo.
2) Attivare l'icona del file facendo comparire la sagoma di colore diverso.
3) Trascinare la sagoma nella parte bassa dello schermo sotto il taccuino e depositarla li premendo il bottone del fuoco.
4) Estrarre dall'unità il floppy del file sorgente.

5) Inserire il nuovo floppy e aprirlo.
6) Trascinare l'icona depositata in basso dentro il taccuino, che ora evidenzia la directory del nuovo floppy, e premere il bottone del fuoco.
7) Seguire le istruzioni del GEOS che chiede di alternare i dischetti sorgente e destinazione.

Tieni presente che i due floppy devono avere nomi diversi, altrimenti le operazioni non vengono eseguite.

Copia di un file tra floppy diversi disponendo di due unità a disco.
Deve essere stata attivata la seconda unità a disco utilizzando il comando DISK, come vedremo più avanti; quindi a destra del taccuino compaiono due icone di floppy, una attiva e l'altra no, ma con i due nomi dei floppy evidenziati.
1) Evidenziare sul taccuino l'icona del file da copiare che sta sul floppy attivo.
2) Attivare l'icona del file facendo comparire la sagoma di colore diverso.
3) Trascinare la sagoma nella parte bassa dello schermo sotto il taccuino e depositarla li premendo il bottone del fuoco.
4) Portare la freccia sull'icona del secondo floppy e premere il bottone del fuoco; così viene attivato il floppy e compare nel taccuino la sua directory. Evidenziare la pagina desiderata della directory.
5) Trascinare l'icona depositata in basso dentro il taccuino, che ora evidenzia la directory del nuovo floppy, e premere il bottone del fuoco.
6) Attendere la fine delle operazioni che vengono eseguite senza intervento dell'operatore.

COME SI STAMPA UN FILE

Partire dal *desktop iniziale*.
La procedura è analoga a quella di cancellazione di un file, solo che invece di trasferire la sagoma del file sul cestino della carta straccia, lo si trasferisce sulla stampante.
Perché possa aver luogo la stampa si deve lavorare con un file stampabile.

PULL DOWN-MENU GEOS

Portare la freccia sulla parola GEOS in alto a sinistra e premere il bottone del fuoco; compare il menu GEOS (vedi figura 2.3) con le seguenti opzioni:
GEOS INFO, DESKTOP INFO, SELECT PRINTER, SELECT INPUT, PREFERENCE MGR, ALARM CLOCK, PHOTO MANAGER, TEXT MANAGER, CALCULATOR, NOTE PAD.
Per scegliere un'opzione si deve far scendere la freccia e premere il bottone del fuoco; altri movimenti impressi alla freccia provocano la disattivazione del menu.

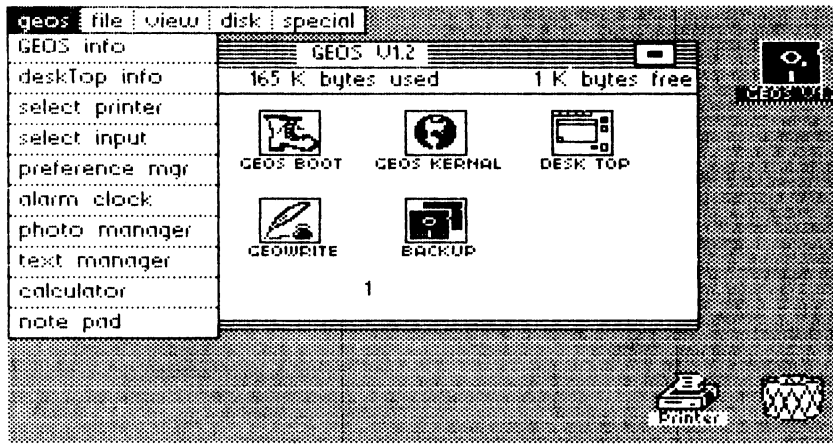


Figura 2.3 PULL DOWN-MENU GEOS

GEOS INFO: mostra i nomi degli autori e il simbolo di copywriter. Premendo il bottone del fuoco scompare la finestra e torna il desktop.

DESKTOP INFO: mostra il nome dell'autore e il simbolo di copywriter. Premendo il bottone del fuoco scompare la finestra e torna il desktop.

SELECT PRINTER: mostra l'elenco delle stampanti per le quali è disponibile il driver sul floppy e consente di selezionare quella che si vuole collegare al calcolatore. Ricorda che il file che contiene il driver della stampante collegata deve essere il primo tra quelli dello stesso tipo presenti nella directory; in consenguenza, quelli che non servono, possono essere cancellati, oppure quello che interessa deve essere spostato e posto prima degli altri.

SELECT INPUT: mostra l'elenco delle device di input; per il momento solo il joystick (o il mouse) e chiede conferma. Portandosi su OK e premendo il bottone del fuoco scompare la finestra sovrapposta.

NOTE PAD: è un *accessorio* che può essere utilizzato da tutte le applicazioni. Esso fornisce un taccuino del quale compare la prima pagina e sul quale si possono scrivere annotazioni da conservare e richiamare; esso è mostrato in figura 2.4.

Sul taccuino si scrive normalmente e si può cancellare con il tasto DEL. Per cambiare pagina si agisce sull'angolo ripiegato in basso a sinistra con le solite modalità. Per conservare quanto scritto e disattivare l'accessorio si agisce sulla finestrella in alto a destra.

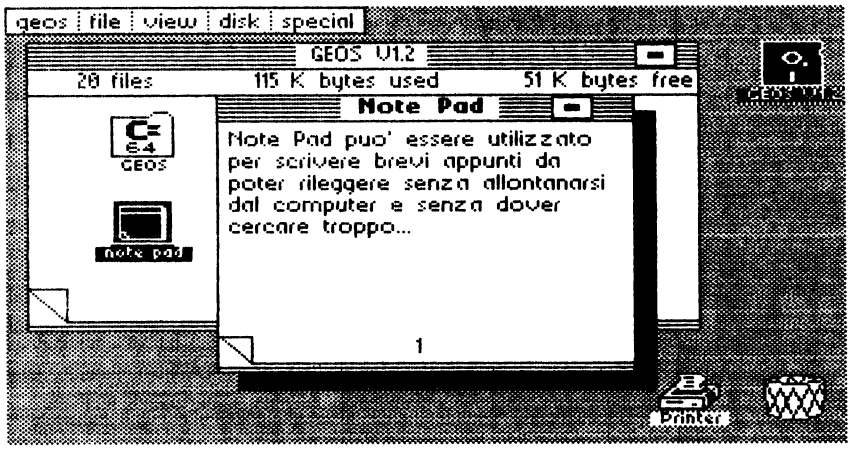


Figura 2.4 Finestra dell'accessorio NOTE PAD

PREFERENCE MGR: è un *accessorio*; esso mostra il quadro riportato in figura 2.5 e consente di modificare alcune caratteristiche dell'ambiente di lavoro GEOS.

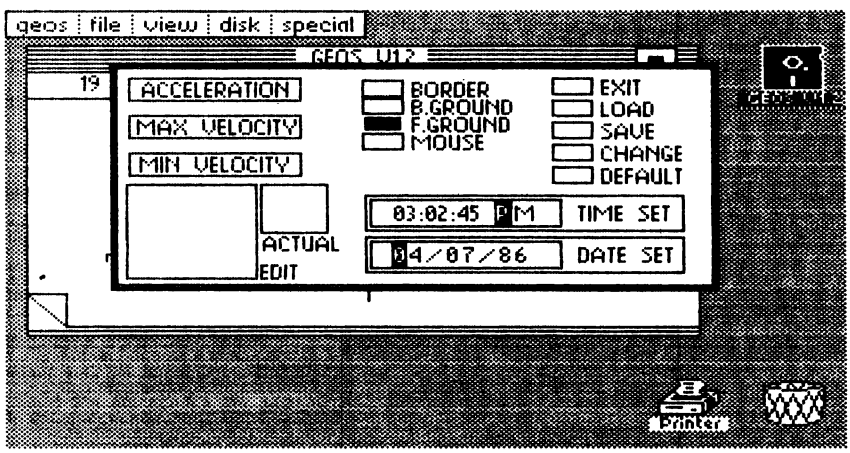


Figura 2.5 Finestra dell'accessorio PREFERENCE MGR

La parte in alto a sinistra riguarda la velocità della freccia-indicatore; possono essere modificate la velocità massima e minima e l'accelerazione portando la freccia sul relativo indicatore, premendo il bottone del fuoco per agganciarlo, spostando poi l'indicatore e premendo nuovamente il bottone del fuoco per fissare la nuova posizione. Per cambiare il colore della freccia, riportata nel riquadro ACTUAL, bisogna portare la freccia nel quadratino colore denominato MOUSE e premere il bottone del fuoco tante volte fino ad ottenere il colore desiderato. Per cambiare la forma della freccia-indicatore bisogna entrare nel riquadro EDIT con la freccia e disegnare o cancellare i singoli pixel premendo il bottone del fuoco.
Nella parte centrale del quadro sono presenti i rettangolini che riportano i colori del bordo, dello sfondo e dei caratteri. Si agisce su di essi come per il colore MOUSE.
Per modificare l'orologio si deve portare la freccia entro il riquadro, scrivere sulla tastiera A per antimeridiano o P per pomeridiano seguito dall'ora e terminare con RETURN.
Per modificare la data si deve portare la freccia nella relativa area nella posizione voluta e scrivere la nuova data da tastiera.
La parte di comandi in alto a destra ha il seguente scopo:
EXIT: fa uscire dal menu di PREFERENCE MGR.
LOAD: richiama dal disco i valori precedenti di tutti i parametri.
SAVE: salva su disco i valori nuovi selezionati per i parametri.
CHANGE: rende attivi tutti i cambiamenti effettuati.
DEFAULT: assegna ad ogni parametro dei valori predefiniti dal sistema.
Per ottenere la conservazione dei nuovi parametri selezionati devi utilizzare il comando CHANGE e il comando SAVE prima di EXIT.

ALARM CLOCK: viene considerato un *accessorio*, può essere usato da tutte le applicazioni e consente di visualizzare e modificare l'ora e di predisporre una sveglia, che suonerà anche se l'accessorio non è attivo. Dopo averlo selezionato compare il quadro video ripportato in figura 2.6.

Per modificare l'ora si deve essere in *modalità orologio*, cioè deve apparire nella prima posizione in basso l'orologio e non la campanella che indica la *modalità sveglia*. Per cambiare modalità si deve posizionare in loco la freccia e premere il bottone del fuoco. Per modificare l'ora si deve agire come in PREFERENCE MGR. Per rendere attiva la scelta si deve portare la freccia su SET e premere il bottone del fuoco.
Ponendo la freccia sul riquadro in basso a destra e premendo il bottone del fuoco si esce dall'accessorio.

PHOTO MANAGER: è un *accessorio* e può essere utilizzato da tutte le applicazioni. Esso consente di realizzare un album di disegni e di gestirlo. Dopo l'attivazione compare una finestra che reca il menu di scelta: CREATE, OPEN, QUIT e sono possibili le tre operazioni:

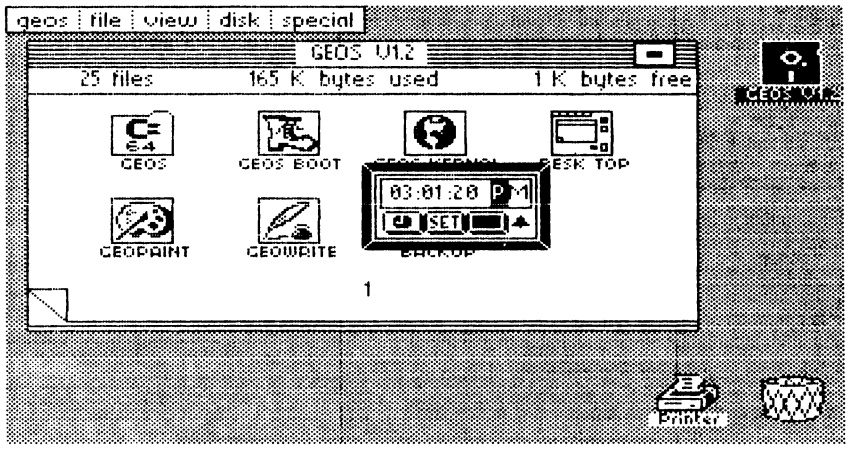


Figura 2.6 Finestra dell'accessorio ALARM CLOCK

1) CREARE EX NOVO L'ALBUM
2) APRIRE L'ALBUM GIÀ ESISTENTE
3) USCIRE DALL'ACCESSORIO.

Dal momento che la creazione dei disegni si attua nell'applicazione GEOPAINT, ritorneremo più avanti su questo accessorio.

TEXT MANAGER: è un *accessorio* utilizzabile da tutte le applicazioni. Esso consente di creare un album di testi e di gestirlo. Dopo l'attivazione si presenta una finestra analoga a quella di PHOTO MANAGER da utilizzare per creare o modificare testi nell'album. L'argomento di questo accessorio verrà ripreso nella spiegazione di GEOWRITER.

CALCULATOR: è un *accessorio* che può essere utilizzato da qualunque applicazione. Dopo l'attivazione appare la finestra riportata in Figura 2.7.

Come si vede si tratta di una normale calcolatrice; si agisce sui tasti per mezzo della freccia e del bottone del fuoco. Per disattivarla si deve agire sul tasto OFF.

Nel menu GEOS compaiono solo i nomi degli accessori che sono presenti sul floppy utilizzato al momento del lancio dell'applicazione in cui compare il menu stesso.

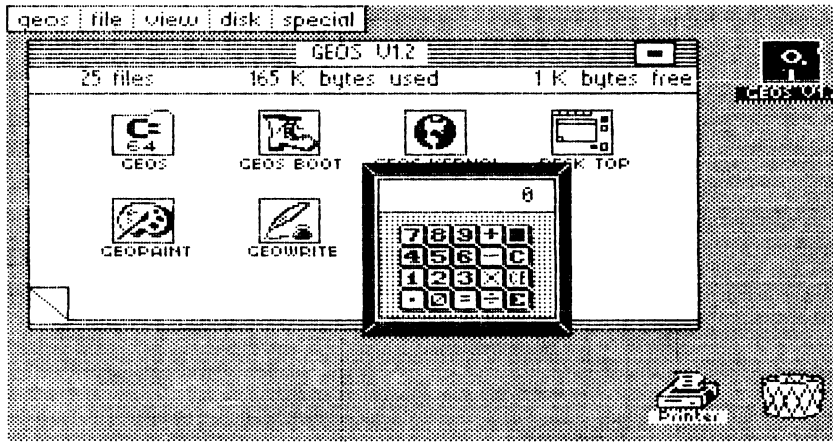


Figura 2.7 Finestra dell'accessorio CALCULATOR

PULL DOWN-MENU FILE

Prima di attivare questo menu è necessario aver selezionato un file presente sulla pagina di taccuino visualizzata.
Dopo l'attivazione compare la finestra sovrapposta riportata in Figura 2.8 e si possono selezionare al solito modo le operazioni: OPEN, DUPLICATE, RENAME, INFO, PRINT.

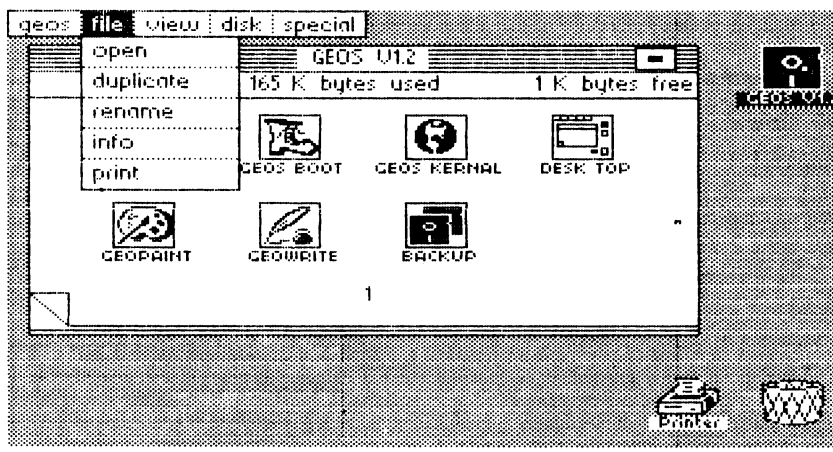


Figura 2.8 PULL DOWN-MENU FILE

OPEN: apre il file già selezionato, cioè ha lo stesso effetto di portare la freccia sull'icona del file e premere due volte il bottone del fuoco. Quello che succede dopo aver aperto un file dipende dal tipo del file selezionato. Per esempio se avevi selezionato NOTE PAD compare sul video il taccuino per le note. Se selezioni un file che contiene un disegno ti viene chiesto di montare un floppy che contenga il programma GEOPAINT.

DUPLICATE: copia il file attivato sullo stesso disco e il GEOS ti chiede il nome da assegnare al file copia.

RENAME: consente di cambiare il nome al file selezionato.

INFO: visualizza le informazioni relative al file selezionato. Tra queste in basso è presente un quadratino con la scritta Write Protect. Se esso è colorato il file è protetto; per togliere la protezione basta portare la freccia sul quadratino e premere il bottone del fuoco. Analogamente se esso non è colorato il file non è protetto e l'operazione determina la protezione del file.
In basso è disponibile una zona dove si possono scrivere delle note che vengono conservate.

PRINT: stampa il file attivato ed equivale a trascinare la sagoma del file sulla printer che si trova in basso a destra sul video.

PULL DOWN-MENU VIEW

Questo menu permette di visualizzare la directory del floppy attivo in diversi modi. Come si vede dalla finestra riportata in figura 2.9 i modi sono: ICON, SIZE, TYPE, DATE, NAME. La scelta si opera nel solito modo.

ICON: propone la situazione del desktop.

SIZE: presenta l'elenco dei file in ordine di grandezza decrescente, riportando: nome, numero Kbyte, tipo file. Sulla finestra compare l'elenco di 8 file; nella parte bassa sono evidenziate due frecce, verso l'alto e verso il basso, e agendo su di esse con il solito sistema si ottiene di far scorrere l'elenco all'indietro o in avanti.

TYPE: come sopra, ma in ordine di tipo.

DATE: come sopra, ma in ordine di data di creazione.

NAME: come sopra, ma in ordine alfabetico in base al nome.

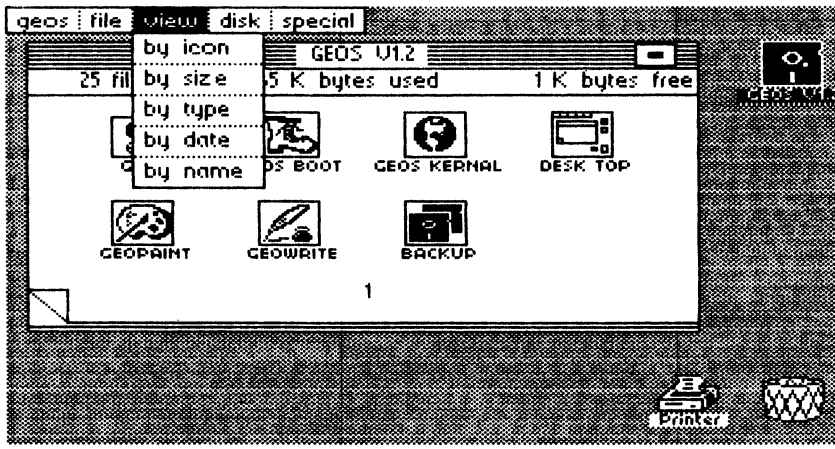


Figura 2.9 PULL DOWN-MENU VIEW

Devi ricordare però che per lavorare sui file e attivarli deve essere presente il desktop con le icone.

PULL DOWN-MENU DISK

Come si può vedere dalla figura 2.10 tale menu riguarda le operazioni disco, e precisamente: OPEN, CLOSE, RENAME, COPY, ADD DRIVE, VALIDATE, FORMAT.

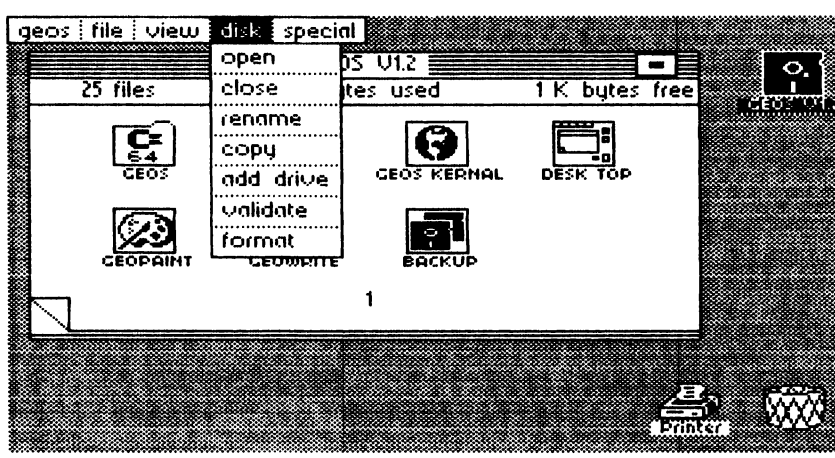


Figura 2.10 PULL DOWN-MENU DISK

OPEN: apre il disco presente, cioè visualizza la prima pagina del menu a icone e il suo nome. Corrisponde a portare la freccia sulla sagoma del dischetto fuori del taccuino e premere il bottone del fuoco.

CLOSE: chiude il disco presente e attivo, cancella le icone dei file, il nome del floppy e pone un punto interrogativo sulla sagoma del dischetto. Corrisponde ad agire sulla finestra posta a destra in alto nel taccuino.

RENAME: consente di cambiare il nome del floppy. Il GEOS chiede il nuovo nome da assegnare.

COPY: permette di eseguire la copia di un floppy. Il dischetto destinazione deve prima essere formattato con l'opzione FORMAT. Durante l'operazione compaiono finestre con i messaggi operativi per lo scambio dei floppy.

ADD DRIVE: consente di aggiungere al sistema una seconda unità a disco. Compaiono finestre con i messaggi operativi necessari per svolgere l'operazione. Al termine a destra del taccuino compare una seconda sagoma di floppy con sotto il nome del dischetto presente. Quando sono presenti due floppy risulta attivo quello selezionato e le operazioni di copia si svolgono in modo diverso.
Nella figura 2.11 è riportato il quadro video che appare con due unità disco collegate.

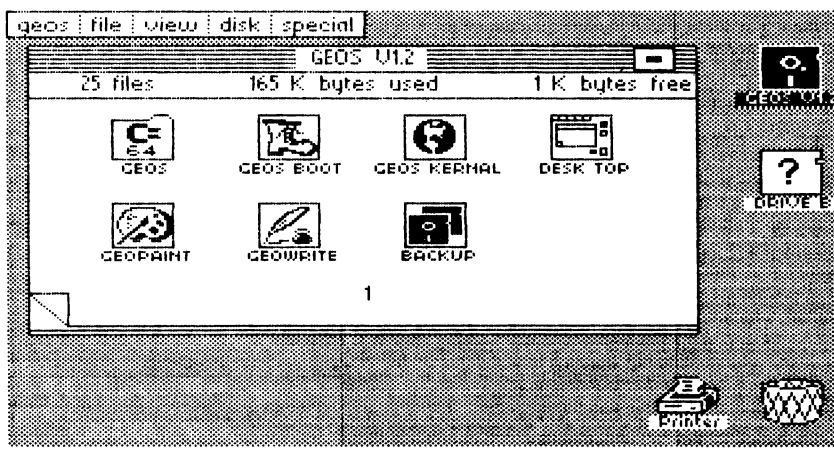


Figura 2.11 Quadro video dopo ADD DISK

VALIDATE: esegue l'operazione di controllo sulla directory del floppy liberando i blocchi che sono occupati in modo non corretto.

FORMAT: consente di formattare un dischetto assegnandogli un nome. Il GEOS chiede il nome da assegnare.

PULL DOWN-MENU SPECIAL

Come riportato in figura 2.12 sono possibili le tre opzioni: BASIC, RESET, Q-LINK.

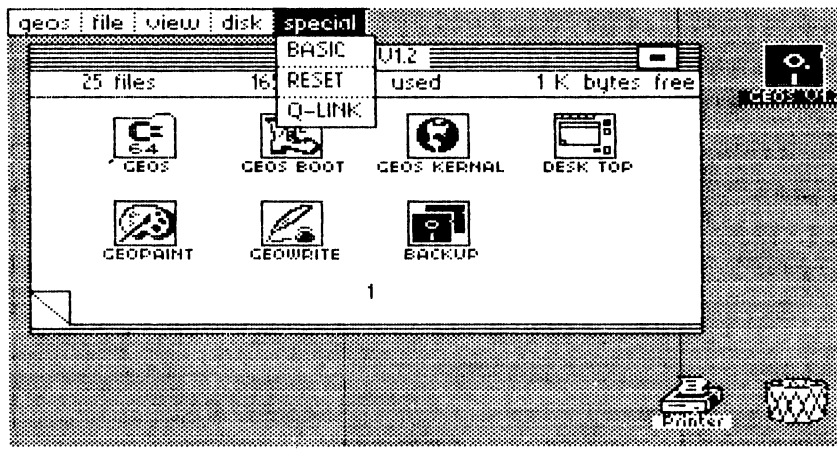


Figura 2.12 PULL DOWN-MENU SPECIAL

BASIC: fa uscire dal GEOS e mostra il ritorno in ambiente BASIC sotto il sistema operativo residente in ROM. Puoi scrivere un programma BASIC e poi rientrare in GEOS lanciandolo nuovamente.

RESET: ricarica il GEOS dal floppy attivo e mostra il desktop iniziale. Se il dischetto attivo non è quello del GEOS, chiede di montarlo.

Q-LINK: mostra una finestra con la richiesta di montare il dischetto Q-LINK, di cui ora non disponiamo.

2.2 APPLICAZIONE GEOPAINT

Si tratta di un pacchetto grafico, che consente di creare disegni, modificarli, memorizzarli su disco in un album, richiamarli e inserire anche parti di testo, scegliendo il carattere in uno dei *font* disponibili.
L'applicazione si seleziona portando la freccia sull'icona del file e premendo due volte il bottone del fuoco. Dopo poco compare il quadro video riportato in figura 2.13.

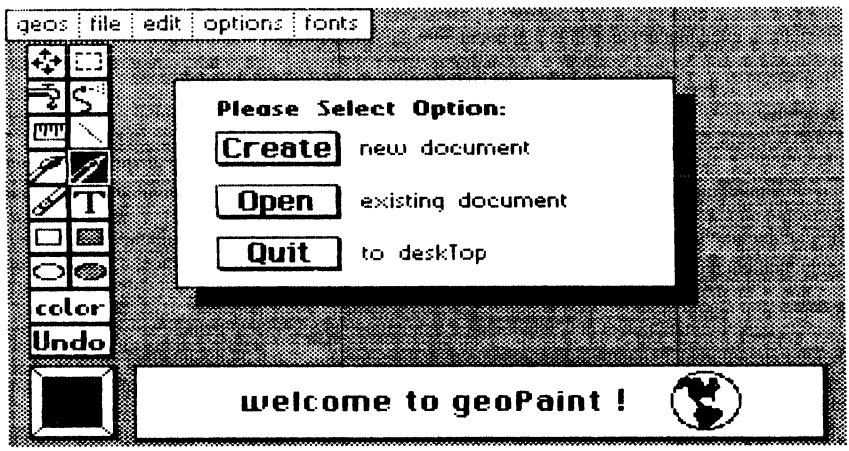


Figura 2.13 Quadro video di GEOPAINT

Come si vede nella parte in alto a sinistra compare un elenco di pull down-menu richiamabili: GEOS, FILE, EDIT, OPTIONS, FONT, da attivare con il solito sistema.

A sinistra compare una striscia contenente diverse icone, che possono essere selezionate una alla volta per lavorare sul foglio da disegno. L'attivazione di una nuova icona comporta la disattivazione di quella attiva precedentemente.

In centro compare una finestra con il menu dei comandi possibili:
- CREATE: per aprire un nuovo foglio da disegno, che corrisponderà sul floppy attivo al file di cui ti viene chiesto il nome;
- OPEN: per aprire un foglio già esistente, compare l'elenco dei file e devi selezionarne uno e aprirlo;
- QUIT: per tornare al GEOS.

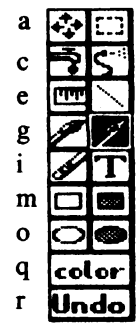
Se sono collegate due unità a disco nei menu secondari compare anche l'opzione per la selezione dell'unità.

I menu riportati in alto a sinistra non sono accessibili fino a quando non si è aperto un documento (foglio da disegno).

Dopo aver aperto un documento, nuovo o preesistente, sul video ne compare una parte; l'intero foglio viene mostrato nella parte bassa del video di fianco alle opzioni possibili per il comando attivo. La freccia-indicatore cambia di forma passando dalle zone comandi, dove mantiene la forma precedente, alla zona grafica.
I comandi per disegnare vengono impartiti servendosi del mouse o del joystick con le solite modalità. Gli strumenti per disegnare, come per esempio la matita o il pennello, quando attivati si appoggiano sul foglio da disegno alla prima pressione del bottone del fuoco e gli spostamenti producono disegno; premendo ulteriormente il bottone del fuoco essi vengono sollevati dalla zona grafica e gli spostamenti successivi non lasciano tracce.

Riportiamo nella figura 2.14 la striscia con le icone delle operazioni eseguibili, visualizzata a sinistra nel video, ponendo a sinistra e a destra il significato delle icone corrispondenti.

a) sposta la finestra grafica
c) riempie di colore
e) righello per misurare
g) pennello
i) gomma per cancellare
m) tracciatore rettangoli vuoti
o) tracciatore cerchi vuoti
q) modifica colori
r) annullamento modifica



b) rettangolo con lati paralleli ai bordi
d) spruzza colore
f) traccia segmenti
h) matita
l) trasferibili testo
n) tracciatore rettangoli pieni
p) tracciatore cerchi pieni

Figura 2.14 Operazioni possibili

L'indicatore grafico cambia forma a seconda dell'opzione selezionata, per esempio, selezionando la gomma per cancellare esso assume una forma rettangolare.
Nella parte bassa dello schermo, dopo aver aperto un documento, compare il foglio da disegno con evidenziata la zona grafica visualizzata, un quadratino-indicatore del colore dei caratteri ed una tavolozza dei colori con una freccia che indica qual'è il

colore selezionato. Se si sposta la freccia che indica il colore e se ne sceglie un altro cambia il colore del quadratino-indicatore e i successivi disegni avranno quel colore. Prendiamo in esame tutte le icone della figura 2.14.

a) Sposta la finestra grafica: se attivato consente di spostare nelle quattro possibili direzioni la finestra grafica; l'effetto può essere controllato sulla parte bassa del video.

b) Rettangolo con lati paralleli ai bordi: consente di definire rettangoli con i lati paralleli alla finestra grafica. I rettangoli sono definiti posizionandosi sull'estremo di una diagonale e premendo il bottone del fuoco e ripetendo l'operazione per l'altro estremo. Quando comincia ad essere visualizzato il rettangolo, cambia il contenuto della parte bassa del video ed appare il menu delle opzioni possibili. Esse sono:

COPY, MOVE, MIRROR X, MIRROR Y, INVERTE, ROTATE, CLEAR.

Per agire sul rettangolo lo devi *agganciare* portando l'indicatore sull'estremo della diagonale e premendo il bottone del fuoco, poi, se hai selezionato COPY o MOVE e sposti l'indicatore, esso si trascina dietro il rettangolo e lo copia o lo sposta nella nuova posizione, fissandolo quando premi il bottone del fuoco.

c) Riempie di colore: è indicato da un rubinetto, che riempie di colore la zona dove si trova l'indicatore grafico fino ai bordi di colore diverso dallo sfondo che incontra, quando si preme il bottone del fuoco.

d) Spruzza colore: spruzza colore nella zona dove si trova l'indicatore grafico quando si preme il bottone del fuoco.

e) Righello per misurare: serve per misurare, in pixel o in pollici, la distanza tra due punti. Devi:
- attivarlo,
- portare l'indicatore sul primo punto e premere il bottone del fuoco,
- portare l'indicatore sul secondo punto e premere il bottone del fuoco,
- in basso compare la misura, in pixel o pollici a seconda dell'opzione attiva, che può essere modificata.

f) Traccia segmenti: dopo averlo attivato devi selezionare un punto con il bottone del fuoco, spostare l'indicatore e arrivato al punto finale premere nuovamente il bottone del fuoco.

g) Pennello: traccia una striscia la cui larghezza dipende dalle opzioni selezionate.

h) Matita: traccia una linea.

i) Gomma per cancellare: cancella dove passa, dopo essere stata attivata.

l) Trasferibili testo: si utilizza per aggiungere parti di testo ad un disegno. Dopo aver attivato l'icona, compare nella parte bassa un menu che consente di scegliere uno stile (FONTS) e il colore. Per poter scrivere, utilizzando la tastiera, devi definire una *finestra di scrittura*; questo lo ottieni postando l'indicatore in una posizione e premendo il bottone del fuoco, poi spostando l'indicatore e premendo nuovamente il bottone del fuoco (cioè indicando gli estremi della diagonale come si fa con l'opzione *b*). Dopo aver definito la finestra, al suo interno è visibile un indicatore (barretta verticale) e puoi scrivere con la tastiera usando RETURN per andare a capo all'interno della finestra. Quando esci dalla finestra e premi il bottone del fuoco scompare la riquadratura della finestra e resta solo il testo.

m) Tracciatore rettangoli vuoti: disegna rettangoli nel colore selezionato definendo la diagonale al solito modo.

n) Tracciatore rettangoli pieni: come sopra ma riempiendoli del colore selezionato.

o) Tracciatore cerchi vuoti: disegna cerchi del colore selezionato. Devi definire il centro e l'estremo del raggio.

p) Tracciatore cerchi pieni: come sopra, ma i cerchi sono pieni di colore.

q) Modifica colori: fa comparire in basso due tavolozze di colori, PAINT per colore disegno e CANVAS per colore sfondo. Può essere scelto un colore per ogni zona.

r) Annullamento modifica: ogni modifica riportata diventa definitiva solo quando ne inizi una nuova; attivando questa icona annulli l'ultima modifica effettuata.

Nella parte bassa del video a sinistra appare un riquadro a fondo scuro uniforme; se porti l'indicatore al suo interno e premi il bottone del fuoco compaiono di fianco diversi disegni, tipo pavimentazione. Portando su uno di questi quadrotti l'indicatore puoi selezionare una pavimentazione premendo il bottone del fuoco. Lo sfondo scelto compare nel riquadro di sinistra. Quando esegui operazioni che comportano un riempimento di colore, esse vengono eseguite con il disegno selezionato.

Esaminiamo ora i *pull down-menu* disponibili in GEOPAINT.

PULL DOWN-MENU GEOS

L'opzione GEOPAINT INFO visualizza alcune informazioni sull'autore e il copywriter. Per il resto il menu è uguale a quello già descritto per il GEOS. Sono disponibili tutti gli accessori già descritti.

PULL DOWN-MENU FILE

Il menu mostra le opzioni: CLOSE, UPDATE, PREVIEW, RECOVER, RENAME, PRINT, QUIT.
CLOSE: chiude il file attivo e ripresenta il menu iniziale di GEOPAINT.
UPDATE: aggiorna il file su disco che contiene il disegno presente sul video. In memoria centrale viene mantenuta solo l'area grafica, mentre tutto il foglio da disegno sta in un file su disco. L'operazione di aggiornamento del file su disco avviene a volte a carico del sistema senza che sia richiesta dall'utente.
PREVIEW: mostra la situazione di tutto il foglio da disegno e compare la scritta OK, che deve essere confermata per poter tornare alla visualizzazione della sola zona grafica.
RECOVER: richiama da disco la situazione precedente cancellando le variazioni presenti solo nell'area grafica; è l'opposto di UPDATE.
RENAME: cambia nome al file attivo.
PRINT: stampa il quadro video.
QUIT: abbandona GEOPAINT.

PULL DOWN-MENU EDIT

Il menu mostra le opzioni: CUT, COPY, PASTE. Per usare queste opzioni devi precedentemente definire attivando l'icona BOX (rettangolo con i bordi paralleli ai lati della finestra grafica) un rettangolo nella finestra video. Dopo aver posizionato il rettangolo, premendo CUT il contenuto del rettangolo viene memorizzato su disco nel file PHOTO SCRAP e cancellato dal video. Se, invece, utilizzi l'opzione COPY, il contenuto del rettangolo viene copiato nel file PHOTO SCRAP, ma senza cancellarlo dal video. Utilizzando PASTE, viene ricopiato nel rettangolo, selezionato sulla finestra, il contenuto del file PHOTO SCRAP.

Il menu EDIT può essere utilizzato anche per conservare i disegni nell'album, attivando l'accessorio PHOTO MANAGER. Infatti puoi attivare tale accessorio dal pull down-menu GEOS; vedrai comparire l'album, con la possibilità di sfogliarlo e il seguente menu:
Create new photo album
Open existing photo album
Quit to desktop.
Dopo aver aperto l'album, selezionando il menu EDIT puoi utilizzare le tre opzioni viste per trasferire il contenuto del file PHOTO SCRAP nell'album e viceversa.

PULL DOWN-MENU OPTIONS

Il menu mostra le opzioni: PIXEL EDIT, NORMAL EDIT, CHANGE BRUSH, COLOR OFF.
PIXEL EDIT: consente di lavorare a livello di *pixel*. Per attivare questa opzione devi posizionare la freccia sulla scritta e premere il bottone del fuoco. Vedrai comparire sul video una riquadratura, che puoi spostare su una zona che ti interessa ingrandire. Raggiunta la posizione voluta, quando premi il bottone del fuoco, la zona grafica completa viene mostrata nella parte bassa del video, mentre nella finestra video compare la zona selezionata ingrandita. Sulla zona ingrandita puoi lavorare con alcuni degli strumenti di GEOPAINT e modificare particolari del disegno. Per tornare alla solita visualizzazione devi selezionare di nuovo il *pull down-menu* e attivare l'opzione NORMAL EDIT.
NORMAL EDIT: serve per uscire dall'opzione PIXEL EDIT e tornare alla visualizzazione normale.
CHANGE BRUSH: serve per modificare le dimensioni del tratto del pennello, selezionandole nella parte bassa del video.
COLOR OFF: selezionandolo scompaiono i colori dal video e il disegno compare in nero. Se dopo averlo selezionato richiami il menu vedi al posto di COLOR OFF la scritta COLOR ON e selezionandola ripristini i colori precedenti.

PULL DOWN-MENU FONTS

Viene presentato un menu che comprende i FONTS di caratteri disponibili: *BSW, CALIFORNIA, CORY, DWINELLE, ROMA, UNIVERSITY. inoltre viene attivata l'icona dei *trasferibili*. Dopo aver scelto il FONTS viene anche chiesto di selezionare la dimensione del carattere. L'asterisco prima del nome indica che quello è il carattere attivo; BSW è attivo per default. Vengono mostrati i primi FONT presenti della directory e tu puoi ordinare i file come vuoi.

2.3 APPLICAZIONE GEOWRITER

Questa applicazione consiste in un programma di *trattamento testi*, che permette di creare testi, conservarli su memoria di massa, richiamarli, modificarli e stamparli. Dopo aver attivato il programma con il solito metodo vedrai comparire il quadro video riportato in figura 2.15.

Il menu presente al centro del quadro di consente di scegliere tra:
Create new document
Open existing document
Quit to desktop.
Se scegli l'opzione Create, ti viene chiesto il nome da assegnare al documento, e devi digitarlo terminando con RETURN.
A questo punto sul video viene mostrata una parte del foglio dove puoi scrivere,

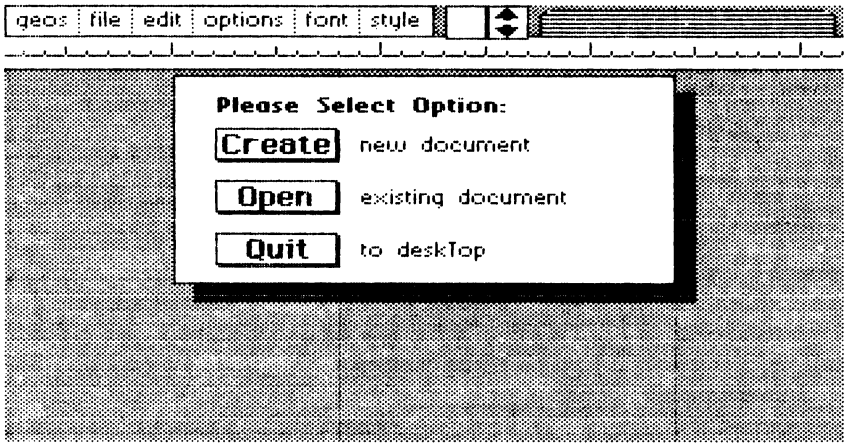


Figura 2.15 Quadro video presentato da GEOWRITER

mentre nella parte alta del video compaiono come sempre a sinistra i nomi dei *pull down-menu* richiamabili e verso il centro un rettangolo, che rappresenta il foglio di scrittura, con un rettangolino, che rappresenta la parte visibile nella finestrea video, e il numero della pagina disponibile. Questo rettangolino può essere agganciato con la pressione del bottone del fuoco e spostato dove vuoi sul foglio, modificando la visualizzazione. Di fianco compaiono due frecce, posizionandosi all'interno di una di esse e azionando il bottone del fuoco puoi spostare il rettangolino di visualizzazione in su o in giù di una linea.
La linea graduata sottostante consente di fissare eventuali tabulatori e di conoscere la posizione dei caratteri.
Quando scrivi una linea che risulta più lunga della parte visualizzata inizialmente, la finestra si sposta automaticamente; premendo RETURN ricompare la finestra iniziale.
Altro modo per ottenere la visualizzazione della parte iniziale del foglio consiste nel muovere tramite il mouse o il joystick la freccia-indicatore verso il margine destro del foglio.
Dal momento che risulta piò gradevole scrivere vedendo costantemente tutta la linea, ti conviene spostare il margine destro, in modo che la linea sia tutta visibile; procedi così:
- sposta la pagina in modo da far comparire la tacca 7 sulla linea graduata;
- aggancia l'indice che sta sotto il 7;
- sposta l'indice dove desideri, per esempio sotto la tacca 5 e fissalo premendo il bottone del fuoco.

Prima di stampare il documento puoi modificare il numero dei caratteri della linea tenendo conto delle caratteristiche della stampante.

Come hai spostato il margine destro del foglio, puoi spostare anche il margine sinistro agendo in modo analogo sull'indice posto sulla tacca 1.
Il *cursore di scrittura* è rappresentato da una barretta verticale; esso si sposta mentre scrivi i caratteri o per effetto del RETURN.
Utilizzando GEOWRITER puoi usare il tasto DEL per *cancellare caratteri.*
Per *inserire caratteri* tra due preesistenti devi postare la freccia-indicatore nel punto desiderato e premere il bottone del fuoco; vedrai comparire il cursore di scrittura in quel punto e viene inserito tutto quello che scrivi, spostando il testo preesistente verso destra.
Per *cancellare parti di testo* devi portare il cursore di scrittura nel punto voluto e poi premere DEL per cancellare.
Le parole che stai scrivendo a fine linea non vengono spezzate, ma trasferite completamente a nuova linea.
Se non intervieni sulla lunghezza della pagina il programma pone 72 linee per pagina.
Per *posizionare tabulatori* devi procedere così:
- portare la freccia-indicatore nel punto voluto sotto la scala graduata e premere il bottone del fuoco; vedrai comparire l'indicazione della fissata tabulazione.
Per raggiungere un punto di tabulazione fissato devi usare i tasti CTRL-I.
Per togliere una indicazione di TAB devi postare la freccia-indicatore sopra il segnalino, agganciarlo e trascinarlo al margine sinistro.
Le opzioni possibili sono: GEOWRITER INFO, PREFERENCE MGR, ALARM CLOCK, PHOTO MANAGER, TEXT MANAGER, CALCULATOR, NOTE PAD.
GEOWRITER INFO fornisce informazione sull'autore e sul copywriter.
Le altre opzioni hanno il significato già visto e rendono disponibili gli *accessori.*

PULL DOWN-MENU FILE

Le opzioni possibili sono: CLOSE, UPDATE, PREVIEW, RECOVER, RENAME, PRINT, QUIT.
Le opzioni hanno il significato già descritto in GEOPAINT, solo che si riferiscono ai documenti di testo e non ai fogli da disegno.

PULL DOWN-MENU EDIT

Le opzioni possibili sono: CUT, COPY, PASTE.
Quando vuoi evidenziare una parte del testo presente sul video devi posizionare il *cursore di scrittura* sul primo carattere, premere il bottone del fuoco e spostare il cursore di scrittura, senza rilasciare il bottone del fuoco, sull'ultimo carattere che ti interessa e rilasciare il bottone del fuoco. La parte di testo selezionato viene evidenziata in campo inverso sul video. Sul testo evidenziato sono possibili diverse operazioni:
- *cancellazione*, premendo il tasto DEL;

- *sostituzione*, puoi scrivere un nuovo testo che va a sostituire quello evidenziato;
- *cambiamento di font o di stile*, puoi selezionare le opzioni FONT e STYLE;
- *trasferimento*, usare CUT e COPY, con i significati già visti in GEOPAINT, per trasferire il testo nel file TEXT SCRAP, e usare PASTE per inserire il contenuto del file TEXT SCRAP in altro punto, dove hai posizionato il cursore. Questa operazione viene guidata da un menu secondario e dà la possibilità di inserire testo o figure create con GEOPAINT.

A questo punto è bene ricordare che con l'accessorio TEXT MANAGER è possibile trasferire il contenuto del file TEXT SCRAP nell'album dei testi e poi servirsi delle pagine dell'album quando servono.

Come vedi in GEOWRITER è possibile spostare e ricopiare paragrafi in punti diversi del testo.

PULL DOWN-MENU OPTIONS

Le opzioni possibili sono: PREVIOUS PAGE, NEXT PAGE, GOTO PAGE, HIDE PICTURES, PAGE BREAK.
Quando il tuo documento è formato da più pagine puoi gestirlo utilizzando queste opzioni.
LAST PAGE: visualizza la pagina precedente.
NEXT PAGE: visualizza la pagina seguente.
GOTO PAGE: chiede il numero della pagina e la visualizza.
HIDE PICTURE: non fa comparire le immagini inserite, ottenendo di aumentare la velocità di visualizzazione.
PAGE BREAK: inserisce un cambio di pagina.

PULL DOWN-MENU FONT

Le opzioni possibili sono: * BSW, CALIFORNIA, CORY, DWINELLE, ROMA, UNIVERSITY.
Consentono di cambiare il FONT di scrittura e la grandezza dei caratteri.

PULL DOWN-MENU STYLE

Le opzioni possibili sono: PLAIN TEXT, BOLD, ITALICS, OUTLINE, UNDERLINE.
Consentono di cambiare lo stile di stampa.

commodore 64
LA GRANDE GUIDA

RITA BONELLI

La Grande Guida del Programmatore permette ai lettori di padroneggiare con soddisfazione il sistema costituito dal Commodore 64, dall'unità a nastro, dall'unità a floppy disk, dalla stampante e dal nuovo adattatore telematico 6499. Inoltre nell'ultima parte viene anche illustrato il nuovo sistema operativo GEOS, sistema a icone, che rende ancora più semplice l'uso del calcolatore.
All'interno dell'Enciclopedia, strumento che risulta essenziale per gli utenti del Commodore 64, si riportano accanto alla trattazione teorica una numerosa serie di programmi esempio, tutti provati sul calcolatore, che consentono di imparare ad usare e programmare il proprio sistema.

● Uno sguardo d'insieme ● Linguaggio BASIC ● Colloquio video/tastiera ● Memorizzazione e caricamento programmi ● I file su stampante ● Costruzione di programmi ● Codici e numeri del calcolatore ● Errori ● Il video e i caratteri ● La tastiera ● Il BASIC compilato ● La programmazione in Assembler e in Linguaggio Macchina ● I file ● File su cassetta ● File su disco ● File sequenziali ● File random ● File relativi ● Magic Desk I ● Calc Result ● Le basi di dati ● La grafica ● Caratteri ● Pagina grafica ● Altre possibilità ● Un piccolo sistema grafico ● Adattatore telematico ● Gli sprite ● Il suono ● I registri del VIC II e i registri del SID ● Utilizzo della memoria e configurazioni ● Il sistema operativo GEOS

GRUPPO EDITORIALE JACKSON

ISBN 88-7056-949-7

9 788870 569490

L. 50.000 Cod. CC749

706

commodore 64
LA GRANDE GUIDA

RITA BONELLI